طراحی الگوریتم- حریصانه

محسن هوشمند

فهرست

# روش حریصانه

آن شنیدستی که در اقصای غور/بارسالاری بیفتاد از ستور

گفت چشم تنگ دنیادوست را/یا قناعت پر کند یا خاک گور

بازرگان هرگز گذشته یا آینده را در نظر نمی‌گیرد. هر روز به دنبال دریافت هرچه‌بیشتر سود است. الگوریتم حریصانه نیز بدین‌گونه عمل می‌کند. در هر لحظه موردی انتخاب می‌شود که بر اساس معیاریْ بهترین به نظر می‌رسد و توجه به انتخاب‌های گذشته و آینده ندارد. عدم توجه به گذشته و آینده و همچنین انتخاب حریصانه در حل بسیاری از مسائل راه‌حل‌هایی کارا و ساده را به دست می‌دهد.

از الگوریتم‌های حریصانه همانند برنامه‌ریزی پویا در حل مسائل بهینه‌سازی استفاده می‌شود. در قیاس با برنامه‌ریزی پویا از خاصیت بازگشتی جهت تقسیم نمونه مسئله به نمونه‌های کوچکتر استفاده می‌شود. اما رهیافت حریصانه تقسیمی در کار نیست و الگوریتم حریصانه با دنباله‌ای از انتخاب‌ها که در لحظه (بدون توجه به گذشته و آینده) بهترین به نظر می‌رسند به پاسخ می‌رسد. در نتیجه، هر الگوریتم به طور موضعی بهینه است به امید اینکه پاسخ سراسری نیز بهینه باشد. ناگفته پیداست که با انتخاب‌هایی که در محل بهینه هستند لزوما به پاسخ بهینه سراسری نخواهیم رسید. در نتیجه در صورتی که روشی حریصانه برای حل مسئله‌ای پیشنهاد شد، نیاز به اثبات بهینگی آن است.

مثال- پرداخت پول به طوری با کمترین تعداد سکه‌ها و برابر مبلغ کل

راه‌حل- در هر زمان

- رویهٔ انتخاب: انتخاب سکه‌ای با بیشترین مبلغ ممکن
- بررسی شدنی‌بودن: که از مبلغ مابه‌التفاوت کمتر باشد.
- بررسی دستیابی به پاسخ: مبلغ جمع شده با کل مبلغ درخواستی برابر شده یا نه

```
While( Vojode Sekke o hal nashodane nemone)
        ENTEKHAB BOZORGTARING Sekke
        If (sekke > mabetafavot)
                Hazf sekke
        Else
                Afzodan sekke
        If ( mblq == mqdar sekkeye jameshode)
                hal
```

با انتخاب سکه‌ای همیشه در پاسخ نهایی باقی خواهد ماند. با رد سکه‌ای با ارزش معین، در ادامه هیچ‌گاه از چنین مقداری استفاده نخواهد شد. در صورتی که مجموعه سکه‌ها شامل ۵۰ تومانی، ۲۵ تومانی، ده تومانی، پنج‌تومانی، یک تومانی باشد پاسخ بهینه به دست می‌آید.

پیش از اشاره به چند مسئلهٔ که راه‌حل حریصانه دارند مفهوم هرم (هیپ) کمینه را یاداوری می‌کنیم که موجب می‌شود بسیاری از مسائل حریصانه با زمان کمتر حل شوند. هرم کمینه درخت دودوئی است که گره والد دارای مقدار یا اندازه کلیدی کمتر نسبت به دو فرزندش است. روش معمول پیاده‌سازی آن آرایه است. به بیان دیگر، اگر مقادیر در آرایه‌ای باشند، آن‌گاه

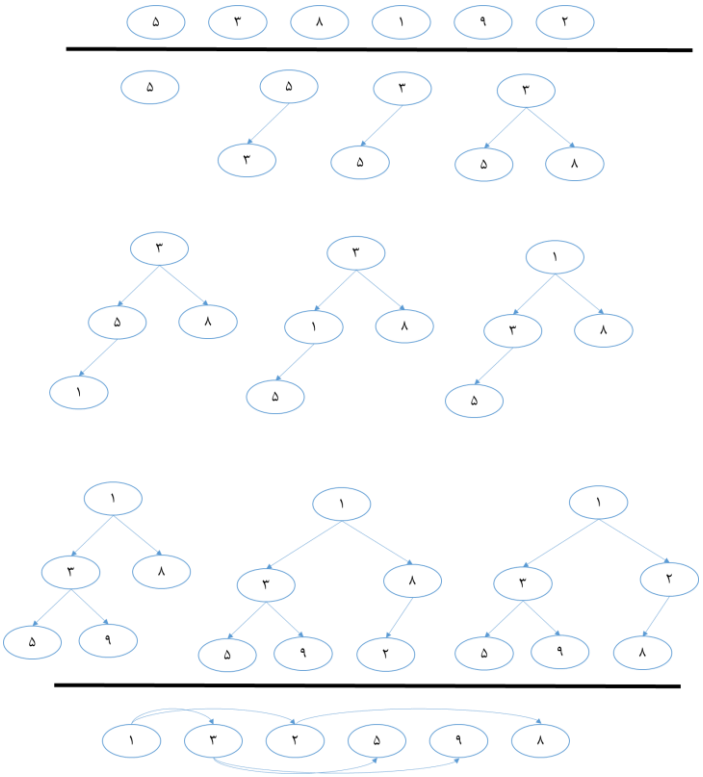
$$\mathrm{Arr}[\mathrm{valed[i]}] \leq \mathrm{Arr[i]},$$

که خاصیت فوق معروف به ویژگی هرم کمینه است. به طریق مشابه، ویژگی هرم بیشینه هم داریم. در آرایه دسترسی به والد و فرزند راست و فرزند چپ به ترتیب به صورت‌های زیر میسر است.

```
Valed[i]
..... return ⌊(i−۱)/۲⌋
frzndChap[i]
..... return ۲i + ۱
frzndRast[i]
..... return ۲i + ۲
```

هرم-کمینه دارای چند عملیات اساسی مانند درج، حذف، بازیابی عضو کمینه است. سخن کوتاه، در پیاده‌سازی هرم کمینه نیاز به رویه‌هایی مانند مقداردهی اولیهٔ هرم، درج اعضای جدید، استخراج عضو کمینه و یا صرفا بازیابی آن، و نمایش هرم است. در واقع، هرم کمینه وابسته به مفهوم دیگری به نام صف‌های اولویت است که اعضای آرایه به ترتیب اولویتی که دارند از صف خارج می‌شوند.

نحوهٔ عملیات آن را با مثالی نمایش می‌دهیم.




فرض می‌کنیم آرایه [5, 3, 8, 1, 9, 2] داده شده است. شکل بالا مراحل عملیات درج و هرم‌سازی را نمایش می‌دهد. ابتدا مقدار پنج در مدخل صفر وارد می‌شود. سپس عدد سه در مدخل ۱ قرار می‌گیرد. به دلیل کوچکتر بودن از سه جابجایی صورت می‌گیرد. ابتدا دو عضو در خانه‌های صفر و یک بررسی می‌شوند. عدد ۸ وارد می‌شود. جابجایی رخ نمی‌دهد. عدد یک به عنوان فرزند چپ عدد ۵ وارد می‌شود. ۵ و ۱ جابجا می‌شوند. ۳ و ۱ نیز جابجا می‌شوند. عدد ۹ به عنوان فرزند سمت راست ۳ وارد می‌شود. جابجایی رخ نمی‌دهد. عدد دو به عنوان فرزند چپ هشت وارد می‌شود. جابجایی صورت می‌گیرد. با جابجایی نهایی عملیات پایان می‌پذیرد.

شبه‌کد ایجاد هرم کمینه به صورت زیر است.

```
Alg. Hramsazi_kmine(arr[], i)
..... kam = i
..... c = chap(i)
..... r = rast(i)
..... if c <= andaze_hram o arr[c] < arr[kam]
..... ..... kam = c
..... if r <= andaze_hram o arr[r] < arr[kam]
..... ..... kam = r
..... if kam ≠ i
..... ..... arr[i] ⟷ arr[kam]
..... ..... Hramsazi_kmine(arr[], kam)
```

زمان اجرا را تحلیل کنید. آیا تابع بالا برای هرم‌سازی آرایه ورودی کافی است. راه‌حلی مطرح کنید.

```
Ijad_hram(arr[])
for i = n/2 -1 : 0:
    Hramsazi_kmine(arr[], i)
```

استخراج از هرم به صورت زیر رخ می‌دهد که عضو نخست برگردانده و عضو آخر به جای آن قرار می‌گیرد و الگوریتم هرم‌سازی کمینه معرفی شده اعمال می‌شود.

```
int estekhrajKmine(arr[])
   n = andaze(arr[])
   if n <= 0
      adam vojod ozv
   if n == 1
      n --
      bazgardandan arr[0]

   rishe = arr[0]
   arr[0] = arr[n - 1];
   n--
   hramsazi(arr[], 0)
   bazgardandan rishe
```

کد سی پیاده‌سازی هرم کمینه به صورت زیر است.

```
#include <stdio.h>
#include <stdlib.h>


typedef struct {
   int andaze;     // tdad aezaye feli dar hearm
   int* gereha; // Aezaye hram
} HeapKmine;


// Tabe swap jhat jabejaee gereha dar hram
void swap(int* a, int* b) {
   int mvqt = *a;
```

```
    *a = *b;
    *b = mvqt;
}


// Tabe yaftan andise valed
int valed(int i) {
    return (i - 1) / 2;
}


// Tabe yaftan andise frznd chap
int frzndChp(int i) {
    return 2 * i + 1;
}


// Tabe yaftan andise frznd rast
int frzndRast(int i) {
    return 2 * i + 2;
}


// Tabe hramsazi zirdrakht nashi gereh i
void hramsazi(HeapKmine* heap, int i) {
    int kochektarin = i;
    int chap = frzndChp(i);
    int rast = frzndRast(i);


    // varasi kochektar bodan frznd chp as kocheketarin
    if (chap < heap->andaze && heap->gereha[chap] < heap->gereha[kochektarin]) {
        kochektarin = chap;
    }


    // varasi kochektar bodan frznd rast as kocktarin
    if (rast < heap->andaze && heap->gereha[rast] < heap->gereha[kochektarin]) {
        kochektarin = rast;
    }


    // Jabejaee dar sorat kochatrin nabdane rishe o edameye hramsazi
    if (kochektarin != i) {
        swap(&heap->gereha[i], &heap->gereha[kochektarin]);
        hramsazi(heap, kochektarin);
    }
}


// Tabe mqdardhi heap ba array vorodi
void Heap_mqdardhi(HeapKmine* heap, int* ArrayVrodi, int Andaze_array) {
    heap->andaze = Andaze_array;
    heap->gereha = (int*)malloc(Andaze_array * sizeof(int));
```

```
    for (int i = 0; i < Andaze_array; i++) {
        heap->gereha[i] = ArrayVrodi[i];
    }


    // Ijad hram-kmine ba hramsazi az gereh valed akhar
    for (int i = (heap->andaze / 2) - 1; i >= 0; i--) {
        hramsazi(heap, i); //heap
    }
}


// Tabe estekhraj ozve kamine az hram o hazfe an
int estekhrajKmine(HeapKmine* heap) {
    if (heap->andaze <= 0) {
        printf("bedon ozve!\n");
        return -1;
    }


    if (heap->andaze == 1) {
        heap->andaze--;
        return heap->gereha[0];
    }


    // bazyabi mqdar kamine o hazf az hram
    int rishe = heap->gereha[0];
    heap->gereha[0] = heap->gereha[heap->andaze - 1];
    heap->andaze--;


    // bazsazi vizhegi hram pas az hazf
    hramsazi(heap, 0);


    return rishe;
}


// Tabe bazyabi mqdar kamine bodone hazf az hram
int daryaftKmine(HeapKmine* heap) {
    if (heap->andaze <= 0) {
        printf("bodon ozve!\n");
        return -1;
    }
    return heap->gereha[0];
}


// Tabe nmayesh gerehaye hram
void nmayeshHeap(HeapKmine* heap) {
    printf("gerehaye hram: ");
    for (int i = 0; i < heap->andaze; i++) {
```

```
        printf("%d ", heap->gereha[i]);
    }
    printf("\n");
}

// Tabe rahasazi hafezeye tkhsisi
void rhasazihram(HeapKmine* heap) {
    free(heap->gereha);
    heap->gereha = NULL;
}

int main() {
    int ArrayVrodi[] = {10, 20, 5, -10, 1, 15, 100, 4, -3, 30, 2};
    int Andaze_array = sizeof(ArrayVrodi) / sizeof(ArrayVrodi[0]);

    HeapKmine heap;
    Heap_mqdardhi(&heap, ArrayVrodi, Andaze_array);

    printf("hram avalliye:\n");
    nmayeshHeap(&heap);

    printf("Ozve kmine: %d\n", daryaftKmine(&heap));

    printf("Estekhraje kmine: %d\n", estekhrajKmine(&heap));
    printf("Heap pasa estekhraj!:\n");
    nmayeshHeap(&heap);

    // rhasazi hafeze
    rhasazihram(&heap);

    return 0;
}
```

با استفاده از هرم می‌توان آرایه‌ای مرتب‌شده تولید کرد.

```
Ijad_hram(arr[])
for I=0:n-1
    estekhrajKmine(arr[])
```

با تغییری جزئی در تابع main می‌توان آرایه‌ای مرتب با ترتیب افزایشی را با هرم کمینه بدست آورد.

```
int main() {
    int ArrayVrodi[] = {10, 20, 5, -10, 1, 15, 100, 4, -3, 30, 2};
    int Andaze_array = sizeof(ArrayVrodi) / sizeof(ArrayVrodi[0]);
```

```
    int Arraykhrodji[Andaze_array];


    HeapKmine heap;
    Heap_mqdardhi(&heap, ArrayVrodi, Andaze_array);


    printf("hram avalliye:\n");
    nmayeshHeap(&heap);


    printf("araye morattab:\n");
    for(int i=0; i < Andaze_array; i++)
    {
       Arraykhrodji[i] = estekhrajKmine(&heap);
       printf("%d ", Arraykhrodji[i]);
    }
    printf("\n");


    // rhasazi hafeze
    rhasazihram(&heap);


    return 0;
}
```

تمرین- مرتبهٔ زمان اجرای مرتب‌سازی اخیر را تحلیل کنید. نحوهٔ اجرای مرتب‌سازی با استفاده از هرم کمینه با روش‌های سریع و ادغامی چه تفاوتی دارد؟

به طریق مشابه، می‌توان هرم بیشنیه را ایجاد کرد و با تغیراتی اندک مرتب‌سازی هرمی کاهشی را با استفاده از هرم بیشینه به دست آورد.

تمرین- شبه‌کد ایجاد آرایه‌ای مرتب با ترتیب کاهشی و افزایشی را با هر دو هرم کمینه و بیشینه حساب کنید.

## مسئله کوله‌پشتی

فرض نگهدارنده‌ای با ظرفیت M و n محصول با ارزش $q_0$ تا $q_{n-1}$ و وزن به ترتیب $vzn_0$ تا $vzn_{n-1}$موجود باشد. اگر بخشی از محصول را در ظرف قرار دهیم، آن‌گاه سود حاصل برابر $q_i * x_i$ خواهد بود. هدف مسئله پرکردن کوله‌پشتی از اجسامی است که سود کلی آنها بیشینه و مجموعه وزن‌ها نیز از ظرفیت کوله‌پشتی بیشتر نباشد. می‌توان مسئله را به صورت زیر تدون کرد.

$$\text{بیش} \sum_{i=0}^{n-1} q_i x_i$$

به شرط

$$\sum_{i=0}^{n-1} vzn_i x_i \le M, 0 \le x_i \le ۱, 0 \le \mathrm{i} \le \mathrm{n} - ۱.$$

پس راه‌حل را می‌توان به صورت n-تایی $(x_0, x_۱, \dots, x_{n-۱})$ است که در دو معادله بالا صدق کنند.

مثال ظرفی با ظرفیت ۲۰ و سه محصول با مشخصات $(vzn_۱, vzn_۲, vzn_۳) = (۱۸,۱۵,۱۰)$ و $(q_۱, q_۲, q_۳) = (۲۵,۲۴,۱۵)$ است. چهار جواب را که از چند روش متفاوت بدست آمده‌اند در جدول زیر آمده است.

| معیار انتخاب | نسبت وزن انتخابی | وزن | ارزش |
|---|---|---|---|
| دلخواه | $\left(\frac{۱}{۲}, \frac{۱}{۳}, \frac{۱}{۴}\right)$ | ۱۶.۵ | ۲۴.۲۵ |
| نزولی ارزش | $\left(۱, \frac{۲}{۱۵}, 0\right)$ | ۲۰ | ۲۸.۲ |
| صعودی وزن | $\left(0, \frac{۲}{۳}, \frac{۱}{۲}\right)$ | ۲۰ | ۳۱ |
| نزولی نسبت ارزش بر وزن | $\left(0, ۱, \frac{۱}{۲}\right)$ | ۲۰ | ۳۱.۵ |

شبه‌کد آن به صورت زیر است.

```
Alg. Kooleposhti_hrisane(vzn[], arzh[], gonjayesh)
..... arzh_be_vzn = arzsh./vzn
..... entekhab mhsolat ba bishtaring mqdare arzsh_be_vzn
..... edame to porshodan koole
..... akharin mhsol baste be gonajyesh baqimande ya lole ya bakhshi az mhsol
```

کوله‌پشتی حریصانه را در سی به صورت زیر می‌توان پیاده کرد.

```
#include <stdio.h>
#include <stdlib.h>


typedef struct {
   int vzn;
   int arzsh;
   double nsbt; // Nesbate Arzesh-be-vazn
} Mhsol;


// Tabe mqayese do mhsol bar asase nsbt arzsh-be-vzn nsbt
int mqayese(const void *a, const void *b) {
   double n1 = ((Mhsol *)a)->nsbt;
   double n2 = ((Mhsol *)b)->nsbt;
```

```
    return (n1 < n2) ? 1 : -1; // tartibe kaheshi
}


//
double kooleposhtiKasri(int zrfiat, Mhsol mhsolha[], int n) {
    // morattabsazi mhsolha bar asase nsbt arzsh-be-vzn ba trtib kaheshi
    qsort(mhsolha, n, sizeof(Mhsol), mqayese);


    double ArzshKol = 0.0; //  arzsh kole mhsolha dar koole
    for (int i = 0; i < n; i++) {
        if (zrfiat >= mhsolha[i].vzn) {
            // ja dadan kole mhsol
            zrfiat -= mhsolha[i].vzn;
            ArzshKol += mhsolha[i].arzsh;
        } else {
            // entekhab bakhshi ya kasri az mhsol
            ArzshKol += mhsolha[i].arzsh * ((double)zrfiat / mhsolha[i].vzn);
            break; // dar sorat por shodan koole
        }
    }


    return ArzshKol;
}


int main() {
    int n = 3; // tdad mhsolha
    int zrfiat = 20; // Zrfiat koole


    // fhrst mhsolha (arzsh, vzn)
    Mhsol mhsolha[] = {
        {18, 25, 0.0},
        {15, 24, 0.0},
        {10, 15, 0.0}
    };


    // mohasebeye nesbate arzsh-be-vzn har mhsol
    for (int i = 0; i < n; i++) {
        mhsolha[i].nsbt = (double)mhsolha[i].arzsh / mhsolha[i].vzn;
    }


    // farakhani tabe jahat hal mas’aleye kooleposhti
    double ArzshBish = kooleposhtiKasri(zrfiat, mhsolha, n);


    printf("bishineye arzsh dar koole = %.2f\n", ArzshBish);


    return 0;
```

```
}
```

پس نسبت $q/vzn$ به نظر پاسخ بهینه را بدست خواهد داد. با در نظر گرفتن مرتب‌سازی مرتبه الگوریتم از $\theta(n\log n)$ است.

می‌توان از هرم بیشینه جهت انجام کار مذکور بهره برد. شبه‌کد آن به صورت زیر خواهد بود.

```
Alg. Kooleposhti_hrisane_hram(vzn[], arzh[], gonjayesh)
..... arzh_be_vzn = arzsh./vzn
..... ijad hram(arzh_be_vzn)
..... estekhrajKmine
..... edame to porshodan koole
..... akharin mhsol baste be gonajyesh baqimande ya lole ya bakhshi az mhsol
```

کد زیر نحوه اجرا را نمایش می‌دهد.

```
#include <stdio.h>
#include <stdlib.h>


// tarif sakhtare mhsol
typedef struct {
   int vzn;
   int arzsh;
   double nsbt;  // Nesbate Arzesh-be-vazn
} Mhsol;


// tarif sakhtare hram kmine
typedef struct {
   int andaze;    // tdad aezaye feli dar hearm
   Mhsol* gereha; // Aezaye hram
} HeapKmine;


// Tabe swap jhat jabejaee gereha dar hram
void swap(Mhsol* a, Mhsol* b) {
   Mhsol temp = *a;
   *a = *b;
   *b = temp;
}


// Tabe yaftan andise valed
int valed(int i) {
   return (i - 1) / 2;
}


// Tabe yaftan andise frznd chap
int frzndChp(int i) {
```

```
    return 2 * i + 1;
}


// Tabe yaftan andise frznd rast
int frzndRast(int i) {
    return 2 * i + 2;
}


// Tabe hramsazi zirdrakht nashi gereh i
void hramsazi(HeapKmine* heap, int i) {
    int bozorgtarin = i;
    int chap = frzndChp(i);
    int rast = frzndRast(i);


    // varasi bozorgtar bodan frznd chp as bozorgtarin
    if (chap < heap->andaze && heap->gereha[chap].nsbt > heap->gereha[bozorgtarin].nsbt) {
        bozorgtarin = chap;
    }


    // varasi bozorgtar bodan frznd rast as bozorgtarin
    if (rast < heap->andaze && heap->gereha[rast].nsbt > heap->gereha[bozorgtarin].nsbt) {
        bozorgtarin = rast;
    }


    // Jabejaee dar sorat kochatrin nabdane rishe o edameye hramsazi
    if (bozorgtarin != i) {
        swap(&heap->gereha[i], &heap->gereha[bozorgtarin]);
        hramsazi(heap, bozorgtarin);
    }
}


// Tabe mqdardhi heap ba array vorodi
void Heap_mqdardhi(HeapKmine* heap, Mhsol* mhsolha, int tdad_mhslha) {
    heap->andaze = tdad_mhslha;
    heap->gereha = (Mhsol*)malloc(tdad_mhslha * sizeof(Mhsol));


    for (int i = 0; i < tdad_mhslha; i++) {
        heap->gereha[i] = mhsolha[i];
    }


    // Ijad hram-kmine ba hramsazi az gereh valed akhar
    for (int i = (heap->andaze / 2) - 1; i >= 0; i--) {
        hramsazi(heap, i); //heap
    }
}
```

```
// Tabe estekhraj ozve kamine az hram o hazfe an
Mhsol estekhrajKmine(HeapKmine* heap) {
   if (heap->andaze <= 0) {
      printf("bodon ozve!\n");
      Mhsol tohi = {0, 0, 0};
      return tohi;
   }


   Mhsol mhsolKmine = heap->gereha[0];
   heap->gereha[0] = heap->gereha[heap->andaze - 1];
   heap->andaze--;


   hramsazi(heap, 0);


   return mhsolKmine;
}


// Kooleposhti-kasri hrisane ba hram-kmineheap
double kooleposhtiKasri(int zrfiat, Mhsol mhsolha[], int n) {
   // Ijad hram-kmine bar asase nsbate arzsh-be-vzn
   HeapKmine heap;
   Heap_mqdardhi(&heap, mhsolha, n);


   double ArzshKol = 0.0; //  arzsh kole mhsolha dar koole


   while (heap.andaze > 0 && zrfiat > 0) {
      Mhsol mhsolFeli = estekhrajKmine(&heap); // estekhraj mhsol ba kamttrin nsbate arzsh-be-
mqdar


      if (zrfiat >= mhsolFeli.vzn) {
         // Afzodan kole mhsol
         zrfiat -= mhsolFeli.vzn;
         ArzshKol += mhsolFeli.arzsh;
      } else {
         // entekhab bakhshi ya kasri az mhsol
         ArzshKol += mhsolFeli.arzsh * ((double)zrfiat / mhsolFeli.vzn);
         break; // dar sorat por shodan koole
      }
   }


   // rahasazi hafeze
   free(heap.gereha);


   return ArzshKol;
}
```

```
int main() {
   int n = 3; // tdad mhsolha
   int zrfiat = 20; // Zrfiat koole

   // fhrst mhsolha (arzsh, vzn)
   Mhsol mhsolha[] = {
      {18, 25, 0.0},
      {15, 24, 0.0},
      {10, 15, 0.0}
   };

   // mohasebeye nesbate arzsh-be-vzn har mhsol
   for (int i = 0; i < n; i++) {
      mhsolha[i].nsbt = (double)mhsolha[i].arzsh / mhsolha[i].vzn;
   }

   // farakhani tabe jahat hal mas'aleye kooleposhti
   double ArzshBish = kooleposhtiKasri(zrfiat, mhsolha, n);

   printf("bishineye arzsh dar koole = %.2f\n", ArzshBish);

   return 0;
}
```

زمان اجرای مسئله کوله‌پشتی کسری با بهره‌گیری از هرم بیشینه (چه تفاوتی با کمینه دارد؟) از مرتبهٔ $nlogn$ است.

قضیه- اگر مسئلهٔ کوله‌پشتی محصولات را بر اساس نسبت ارزش بر وزن متناظر به صورت نزولی مرتب کنیم، آن‌گاه الگوریتم حریصانه معرفی شده پاسخی بهینه برای هر نمونه خواهد یافت.

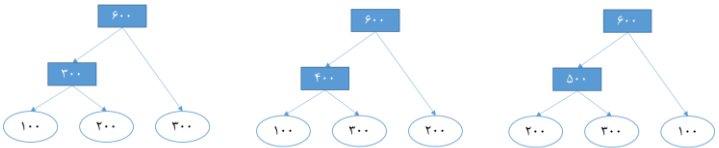
اثبات؟- فرض می‌کنیم که در انتخاب حریصانه از محصول i-ام به نسبت $\mathrm{vzn_i}$ انتخاب کردیم. برهان خلف- حل فرض می‌کنیم در پاسخ بهینه واقعی وزن i فرق کند. حال فرض کنید که کوله پر است و نمی‌توان بیشتر از مقدار فعلی i بدان افزود. پس دنبال محصول j می‌گردیم که $\frac{q_j}{vzn_j} < \frac{q_i}{vzn_i}$. می‌توان محصول j را به میزان وزن x از کوله کم کرد و محصول i را به همان وزن به جعبه افزود. میزان تغییر ارزش کوله برابر با $x.\frac{q_i}{vzn_i} - x.\frac{q_j}{vzn_j} = x.\left(\frac{q_i}{vzn_i} - \frac{q_j}{vzn_j}\right) > 0$ چرا؟ چون $\frac{q_j}{vzn_j} < \frac{q_i}{vzn_i}$ پس حکم ثابت است.

## ادغام دودوئی و بهینه آرایه‌ها

تمرین- دو آرایه مرتب شده با اندازه‌های m و n را در چه زمانی می‌توان ادغام کرد؟ $\theta(m+n)$.

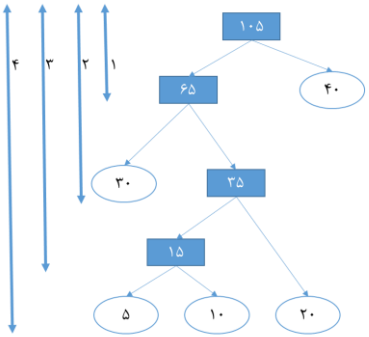
در صورتی که بیشتر از دو آرایه داشته باشیم. می‌توان آرایه واحد را با ترکیب‌های دودوئی متفاوتی بدست آورد.

مثال سه آرایهٔ ۱۰۰ و ۲۰۰ و ۳۰۰ عضوی را داریم ترکیب آنها به آرایه‌ای ۶۰۰ عضوی ختم می‌شود. اما این‌کار را به سه روش متفاوت می‌توان انجام داد. شکل زیر سه روش را نمایش می‌دهد.




ج- جمع مقایسه‌ها ۹۰۰ ب- جمع مقایسه‌ها ۱۰۰۰ الف- جمع مقایسه‌ها ۱۱۰۰

تمرین- پنج آرایه مرتب با اندازه‌های ۲۰، ۳۰، ۱۰، ۵، ۴۰ داریم. حداقل تعداد مقایسه‌ها برای ادغام نهایی آنها چقدر است؟ ۲۲۰




جمع مقادیر داخل مربع‌ها معادل کل تعداد مقایسه‌ها است.

ارتفاع از ریشه را در شکل می‌بینیم. تعداد مقایسه‌ها را به نحو دیگر می‌توان نشان داد

$$۵+۱۰$$

$$(۵+۱۰)+۲۰$$

$$۳۰ + (۵ + ۱۰ + ۲۰)$$

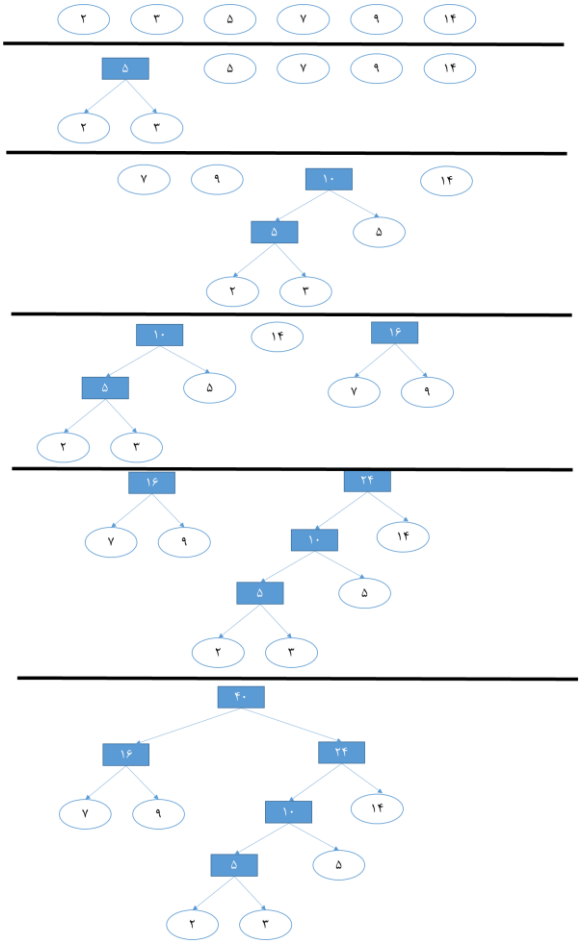
$$(۳۰ + ۵ + ۱۰ + ۲۰) + ۴۰$$

---

$$= ۴ \times ۵ + ۲ \times ۳۰ + ۴ \times ۱۰ + ۳ \times ۲۰ + ۱ \times ۴۰ = ۲۲۰$$

با اندکی بررسی می‌توان دید که کل مقایسه‌ها از معادلهٔ $\sum_{i=0}^{n-1} d_i q_i$ به دست می‌آید به طوری که $q_i$ اندازهٔ آرایه $i$ و $d_i$ عمق یا فاصله از ریشه گره $i$ است. به نوعی میزان وزن مسیر داخلی باید کمینه شود.

```
Alg. Edqam Behine(Arr[], n)
        For i=[0:n-2]
                Gereh(D)
                D->farzandchap = kam(Arr[])
                D->farzandrast = kam2(Arr[])
                Vzn(D) = Vzn(D->farzandchap) + Vzn(D->farzandrast)
                Afzodan (D, Arr[])
```




شبه‌کد آن با استفاده از هرم کمینه به صورت زیر است.

```
Alg. Eqdam Behine(Arr[], n)
..... ijad_hram(Arr[], n)
..... while n > 1
.......... kam1 = estekhrajKmine(Arr[], n)
.......... kam2 = estekhrajKmine(Arr[], n)
.......... mvqt = kam1 + kam2
.......... Afzodan (Arr[], mvqt, n)
```

کد سی مسئلهٔ ادغام بهینه آرایه‌ها به صورت زیر است.

```
#include <stdio.h>
#include <stdlib.h>


// sakhtar hram-kmine
typedef struct {
   int andaze;  // tdad aezaye feli dar hearm
   int* gereha; // Aezaye hram
} HeapKmine;


// Tabe swap jhat jabejaee gereha dar hram
void swap(int* a, int* b) {
   int mvqt = *a;
   *a = *b;
   *b = mvqt;
}


// Tabe yaftan andise valed
int valed(int i) {
   return (i - 1) / 2;
}


// Tabe yaftan andise frznd chap
int frzndChp(int i) {
   return 2 * i + 1;
}


// Tabe yaftan andise frznd rast
int frzndRast(int i) {
   return 2 * i + 2;
}


// Tabe hramsazi zirdrakht nashi gereh i
void hramsazi(HeapKmine* heap, int i) {
   int kochektarin = i;
   int chap = frzndChp(i);
   int rast = frzndRast(i);


   // varasi kochektar bodan frznd chp as kocheketarin
   if (chap < heap->andaze && heap->gereha[chap] < heap->gereha[kochektarin]) {
      kochektarin = chap;
   }


   // varasi kochektar bodan frznd rast as kocktarin
   if (rast < heap->andaze && heap->gereha[rast] < heap->gereha[kochektarin]) {
```

```
        kochektarin = rast;
    }


    // Jabejaee dar sorat kochatrin nabdane rishe o edameye hramsazi
    if (kochektarin != i) {
        swap(&heap->gereha[i], &heap->gereha[kochektarin]);
        hramsazi(heap, kochektarin);
    }
}


// Tabe mqdardhi heap ba array vorodi
void hram_mqdardhi(HeapKmine* heap, int* ArrayVrodi, int Andaze_array) {
    heap->andaze = Andaze_array;
    heap->gereha = (int*)malloc(Andaze_array * sizeof(int));
    for (int i = 0; i < Andaze_array; i++) {
        heap->gereha[i] = ArrayVrodi[i];
    }


    // Ijad hram-kmine ba hramsazi az gereh valed akhar
    for (int i = (heap->andaze / 2) - 1; i >= 0; i--) {
        hramsazi(heap, i);
    }
}


// Tabe estekhraj ozve kamine az hram o hazfe an
int estekhrajKmine(HeapKmine* heap) {
    if (heap->andaze <= 0) {
        printf("Heap is empty!\n");
        return -1;
    }


    if (heap->andaze == 1) {
        heap->andaze--;
        return heap->gereha[0];
    }


    // bazyabi mqdar kamine o hazf az hram
    int rishe = heap->gereha[0];
    heap->gereha[0] = heap->gereha[heap->andaze - 1];
    heap->andaze--;


    // bazsazi vizhegi hram pas az hazf
    hramsazi(heap, 0);


    return rishe;
}
```

```
// Darj ozve jdid dar hram
void AfzodanOzve(HeapKmine* heap, int arzesh) {
   heap->andaze++;
   heap->gereha = (int*)realloc(heap->gereha, heap->andaze * sizeof(int));
   int i = heap->andaze - 1;
   heap->gereha[i] = arzesh;
   while (i != 0 && heap->gereha[valed(i)] > heap->gereha[i]) {
      swap(&heap->gereha[i], &heap->gereha[valed(i)]);
      i = valed(i);
   }
}


// Tabe nmayesh gerehaye hram
void nmayeshHeap(HeapKmine* heap) {
   printf("Hram: ");
   for (int i = 0; i < heap->andaze; i++) {
      printf("%d ", heap->gereha[i]);
   }
   printf("\n");
}


// Tabe rahasazi hafezeye tkhsisi
void rhasazihram(HeapKmine* heap) {
   free(heap->gereha);
   heap->gereha = NULL;
}


// Edqam Arrayeha
int EdgamArrayha(int* arrayha, int n) {
   HeapKmine heap;
   hram_mqdardhi(&heap, arrayha, n);


   int hazineKol = 0;


   while (heap.andaze > 1) {
      // Estekhraj do arraye kochak az arrayha
      int aval = estekhrajKmine(&heap);
      int dovom = estekhrajKmine(&heap);


      // edqam
      int hazineEdqam = aval + dovom;
      hazineKol += hazineEdqam;


      // Afzodan arraye hasel be arrayha
      AfzodanOzve(&heap, hazineEdqam);
```

```
    }

    rhasazihram(&heap);
    return hazineKol;
}

int main() {
    int arrayha[] = {40, 5, 20, 10, 30};
    int n = sizeof(arrayha) / sizeof(arrayha[0]);

    int hazineKol = EdgamArrayha(arrayha, n);
    printf("Hazine kole edqam arrayha: %d\n", hazineKol);

    return 0;
}
```

زمان اجرا از مرتبه $O(k + nlogk)$ است. به طوری که $k$ تعداد آرایه‌های اولیه و $n$ برابر جمع تمامی اعضاست.

لم- درخت دودوئی ادغام بهینه دارای دو آرایه همزاد با کمترین تعداد است. اثبات آن را به عنوان تمرین کنار می‌گذاریم.

قضیه-اگر فهرستی دارای n راس با وزن‌های $v_0$ تا $v_{n-1}$ باشد، الگوریتم درخت دودویی ادغام بهینه را تولید خواهد کرد.

اثبات- استقرا- تمرین

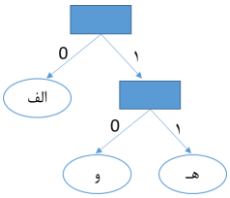
اگر n=1 بدون گره داخلی و در نتیجه بهینه است.

تمرین- ادغام k-تایی

## کد هافمن

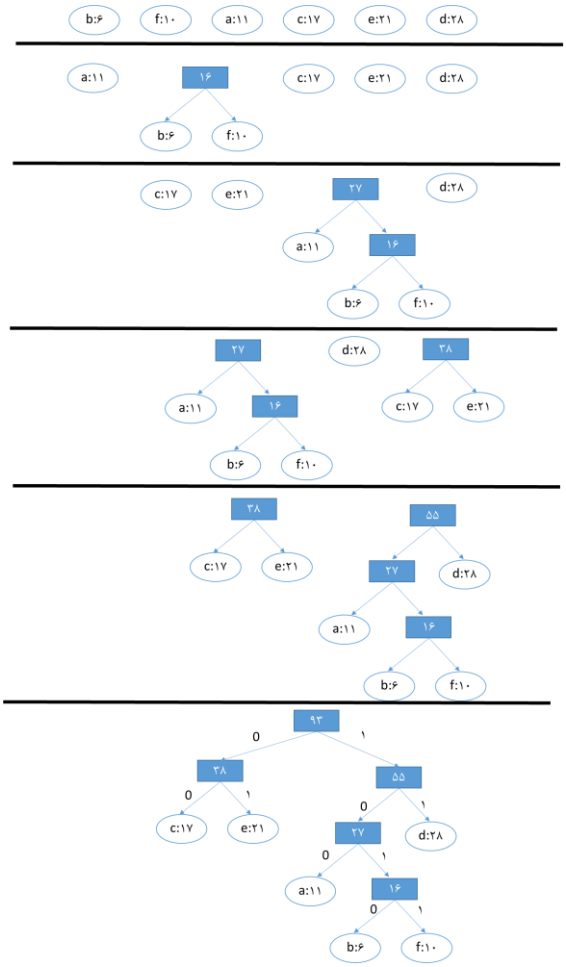
ذخیره کارامد فایل‌ها و داده‌ه یکی از نیزاهای اساسی محاسباتی است. معمولا از فضرده‌سازی داده جهت یافتن رفت تییر و نگهداری کارامد و ارسال بهینه‌تر داده‌استفاده می‌شود یک یاز روش‌ها کد هافمن است که الگوریتمی حریصانه جهت یافتن شیوه نحوه کدگذاری داده است. معمولا داده با الفبای بانمایش با طول یکسان (طول ثابت) تعریف می‌شود. به عنوان مثال الفبا شامل سه حرف است و با مقادیر 00 و 01 و 10 نمایشد اده می‌شود. در نتیجه کد متناظر a: 00 b: 01 c: 11 جهت نمایش ababcbbbc از

۰۰۰۱۰۰۰۱۱۱۰۱۰۱۰۱۱۱ استفاده می‌شود. اما کدهای طول متغیر می‌توان به کارایی بالاتری در مصرف منابع رسید. هافمن از چنین نوع کدهایی است.




تمرین تعداد بیت‌ها برای نمایش جمله مذکور با نمایش بالا چه تعداد است؟

فرض پیشوندی را تحقیق کنید. تعداد بیت‌ها در درخت دودوئی برابر با $bit(T) = \sum_{i=۱}^{n} f(v_i)omq(v_i)$ است.




شبه کد آن با استفاده از هرم بهینه به صورت زیر است

```
Alg. Huffman(arr.nviseha[], arr.fravani[], n)
….. Ijad hram(arr[])
….. while n > 1
……. kam1 = Estekhrajkmine(arr)
```

```
……. kam2 = Estekhrajkmine(arr)
……. grehe jadid.fravani = kam1.fravani + kam.fravani
……. jadid.chpa = kam1; jadid.rast = kam2
……. Afzdan(arr,  jadid)
….. rishe = Estekhrajkmine(arr)
```

کد سی آن به صورت زیر است.

```
#include <stdio.h>
#include <stdlib.h>

// sakhtar gereh derakht Huffmann
typedef struct Gereh {
   char nvise;       // nvise
   int fravani;      // fravani of the nvise
   struct Gereh* chap;   // frzand chap
   struct Gereh* rast;   // frzand rast
} Gereh;

// sakhtar hram-kmine
typedef struct {
   int andaze;         // tdad aezaye feli dar hearm
   Gereh** gereha;      // Aezaye hram
} HeapKmine;

// Tabe swap jhat jabejaee gereha dar hram
void swap(Gereh** a, Gereh** b) {
   Gereh* mvqt = *a;
   *a = *b;
   *b = mvqt;
}

// Tabe yaftan andise valed
int valed(int i) {
   return (i - 1) / 2;
}

// Tabe yaftan andise frznd chap
int frzndChp(int i) {
   return 2 * i + 1;
}

// Tabe yaftan andise frznd rast
int frzndRast(int i) {
   return 2 * i + 2;
}

// Tabe hramsazi zirdrakht nashi gereh i
void hramsazi(HeapKmine* heap, int i) {
   int kochektarin = i;
   int chap = frzndChp(i);
```

```
    int rast = frzndRast(i);

    if (chap < heap->andaze && heap->gereha[chap]->fravani < heap->gereha[kochektarin]->fravani) {
        kochektarin = chap;
    }

    if (rast < heap->andaze && heap->gereha[rast]->fravani < heap->gereha[kochektarin]->fravani) {
        kochektarin = rast;
    }

    if (kochektarin != i) {
        swap(&heap->gereha[i], &heap->gereha[kochektarin]);
        hramsazi(heap, kochektarin);
    }
}

// Tabe mqdardhi heap ba array vorodi
void Heap_mqdardhi(HeapKmine* heap, Gereh** Gereha, int tdad_Gereha) {
    heap->andaze = tdad_Gereha;
    heap->gereha = (Gereh**)malloc(tdad_Gereha * sizeof(Gereh*));
    for (int i = 0; i < tdad_Gereha; i++) {
        heap->gereha[i] = Gereha[i];
    }

                // Ijad hram-kmine ba hramsazi az gereh valed akhar
    for (int i = (heap->andaze / 2) - 1; i >= 0; i--) {
        hramsazi(heap, i);
    }
}

// Tabe estekhraj ozve kamine az hram o hazfe an
Gereh* estekhrajKmine(HeapKmine* heap) {
    if (heap->andaze <= 0) {
        printf("bedon ozve!\n");
        return NULL;
    }

    if (heap->andaze == 1) {
        heap->andaze--;
        return heap->gereha[0];
    }

                // bazyabi mqdar kamine o hazf az hram
    Gereh* rishe = heap->gereha[0];
    heap->gereha[0] = heap->gereha[heap->andaze - 1];
    heap->andaze--;

                // bazsazi vizhegi hram pas az hazf
```

```
    hramsazi(heap, 0);

    return rishe;
}

// Afzodan Gereh jadid be hram
void AfzodanGereh(HeapKmine* heap, Gereh* Gereh) {
    heap->andaze++;
    heap->gereha = (Gereh**)realloc(heap->gereha, heap->andaze * sizeof(Gereh*));
    int i = heap->andaze - 1;
    heap->gereha[i] = Gereh;

    while (i != 0 && heap->gereha[valed(i)]->fravani > heap->gereha[i]->fravani) {
        swap(&heap->gereha[i], &heap->gereha[valed(i)]);
        i = valed(i);
    }
}

// Ijad gereh jadid
Gereh* IjadGereh(char nvise, int fravani) {
    Gereh* mvqt = (Gereh*)malloc(sizeof(Gereh));
    mvqt->nvise = nvise;
    mvqt->fravani = fravani;
    mvqt->chap = mvqt->rast = NULL;
    return mvqt;
}

// Ijad drakht huffman
Gereh* TolidHuffman(char nviseha[], int fravaniha[], int n) {
    HeapKmine heap;
    Gereh* Gereha[n];
    for (int i = 0; i < n; i++) {
        Gereha[i] = IjadGereh(nviseha[i], fravaniha[i]);
    }
    Heap_mqdardhi(&heap, Gereha, n);

    while (heap.andaze > 1) {
        // Estekhraje do gereh ba kmatrin fravani
        Gereh* nkhst = estekhrajKmine(&heap);
        Gereh* dovom = estekhrajKmine(&heap);

        // Ijad gereh dakheli jadid ba estefade az do frzand
        Gereh* jadid = IjadGereh('$', nkhst->fravani + dovom->fravani);
        jadid->chap = nkhst;
        jadid->rast = dovom;

        // Darje gereh jdid be hram
        AfzodanGereh(&heap, jadid);
    }
```

```
    // Greeh baqimande risheye drakhte Huffman
    Gereh* rishe = estekhrajKmine(&heap);
    free(heap.gereha);
    return rishe;
}

// nemayesh bazgashticodehaye Huffman
void nmayeshCodeha(Gereh* rishe, int code[], int balatrin) {
    if (rishe->chap) {
        code[balatrin] = 0;
        nmayeshCodeha(rishe->chap, code, balatrin + 1);
    }

    if (rishe->rast) {
        code[balatrin] = 1;
        nmayeshCodeha(rishe->rast, code, balatrin + 1);
    }

    if (!rishe->chap && !rishe->rast) {
        printf("%c: ", rishe->nvise);
        for (int i = 0; i < balatrin; i++) {
            printf("%d", code[i]);
        }
        printf("\n");
    }
}

int main() {
    char nviseha[] = {'a', 'b', 'c', 'd', 'e', 'f'};
    int fravaniha[] = {5, 9, 12, 13, 16, 45};
    int n = sizeof(nviseha) / sizeof(nviseha[0]);

    Gereh* rishe = TolidHuffman(nviseha, fravaniha, n);

    int code[100], balatrin = 0;
    printf("codehaye Huffman:\n");
    nmayeshCodeha(rishe, code, balatrin);

    return 0;
}
```

## مسئله زمان‌بندی

زمان انتظار و زمان ملاقات: زمان در سامانه

راه‌حل کمینه کردن زمان که در سیستم در نظر گرفتن تمامی زمان‌بندی‌ها و اجرای کمترین مقدار

مثال- $t1 = ۶$، $t2 = ۱۰$، $t3 = ۵$ پرانتز زمان اجرا و قلاب زمان انتظار را نمایش می‌دهد. زمان کل در سیستم ماندن برابر با جمع مقادیر است.

| فعالیت | زمان در سیستم |
|---|---|
| ۱ | $[0] + (۶)$ |
| ۲ | $[۶] + (۱۰)$ |
| ۳ | $[۶ + ۱۰] + (۵)$ |

جمع کل زمان ماندن در سامانه برابر ۴۳ است.

حال اگر تمامی حالات را بسنجیم حالت بهینه $[۳,۱,۲]$ یا زمان کل ۳۷ است. بررسی تمامی حالت از مرتبهٔ فاکتوریل است. اما در صورت انتخاب با کمترین زمان اجرا حالت بهینه بدست می‌اید. شبه‌کد مسئله به صورت زیر است

مرتب‌سازی صعودی فعالیت‌ها بر اساس زمان اجرا
تا زمان حل نشدن تمامی فعالیت‌ها
زمان‌بندی فعالیت بعدی
حل شدن فعالیت مذکور

تمرین- مرتبه زمانی را به دقت بررسی کنید.

قضیه- انتخاب فعالیت‌ها بر اساس زمان اجرای صعودی زمان‌بندی بهینه است.

اثبات. فرض کنید $T$ زمان اجرای روش صعودی باشد. حال زمانبندی غیرصعودی را فرض می‌کنیم که زمان کمتری دارد$T'$. به دلیل مرتب نبودن زمابندی غیرصعودی، موردی وجود دارد که زمان اجرای فعالیت $i$ از زمان اجرای فعالیت بعدی بیشتر است. حال زمان‌بندی $T'$ را در نظر می‌گیریم که جای دو فعالیت مذکور را عوض کند. پس داریم

$$T' = T + t_{i+1} - t_i$$

به دلیل $t_i > t_{i+1}$ پس $T' < T$، در نتیجه برهان خلف غلط او حکم ثابت ست.

مسائل مشابه دیگری وجود دارد که دارای قیدهای بیشتری اعم از مهلت اجرا، ارزش هر کار، و موارد دیگر هستند که روش‌های حریصانه برای آنها معرفی می‌شود. البته همیشه بهترین پاسخ ممکن را نمی‌یابد. مثال زیر را ببینید.

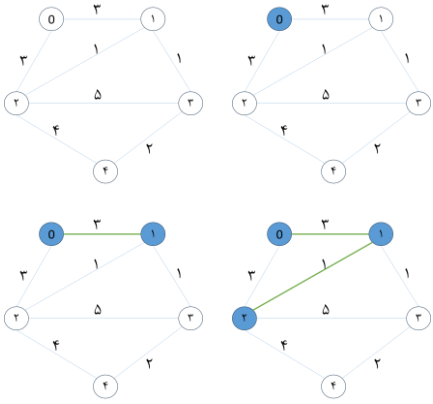
## درخت پوشای کمینه

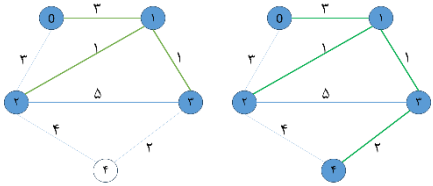
گراف بی‌جهت را با وزن‌های نامنفی فرض داریم. مسیر در گراف بی‌جهت عبارت از دنباله‌ای از راس‌ها است که هر جفت راس پشت سر هم دارای یال باشند. گراف بی‌جهت را هم‌بند خوانیم اگر بین هر جفت راس مسیری وجود داشته باشد. دور ساده مسیری از راسی به خودش است به طوری که هیچ راس دیگری دو بار در مسیر نباشد. گراف بی‌دور دارای هیچ دوری نیست. درختْ گرافِ بی‌جهتِ هم‌بندِ بی‌دور است. درخت پوشای گراف زیرگرافی است که دارای همه راس‌ها باشند و درخت باشد. درخت پوشای کمینه درخت پوشایی با حداقل جمع وزن یال‌ها است.

در صورت استفاده از جستجوی کامل درخت پوشای کمینه، مسئله دارای بدترین زمان از مرتبه نمایی است. اما امکان حل کارآمدتر مسئله با نگاهی حریصانه وجود دارد.

### پریم

یکی از روش‌های حریصانه الگوریتم پریم است. در ابتدا مجموعه‌ای تهی را بابت افزودن یال‌های درخت پوشای کمینه تعریف می‌کنیم. نام آن را F می‌گذاریم. مجموعه تهی Y را برای راس‌ها در نظر می‌گیریم. مجموعه راس‌ها Y را برابر راس نخست قرار می‌دهیم. سپس نزدیک‌ترین راس به راس موجود را از مجموعه V-Y می‌یابیم و آن را به مجموعه می‌افزاییم همچنین یال مابین را به مجموعه یال‌ها اضافه می‌کنیم. فرایند را تا برابر شدن مجموعه راس‌ها با مجموعه کل راس‌ها ادامه می‌دهیم. سوال- دور را چگونه حل کنیم؟ انتخاب از مجموعه V-Y ضامن در دور نیفتادن است.






جهت پیاده‌سازی ماتریس وزن‌ها را به صورت زیر تعریف می‌کنیم.

$$v_{ij} = \begin{cases} \text{وزن یال موجود} \\ 0, \text{عدم وجود یال} \end{cases}$$

پس ماتریس مجاورت را به صورت زیر تعریف می‌کنیم.

$$\begin{bmatrix} 0 & ۳ & ۳ & 0 & 0 \\ ۳ & 0 & ۱ & ۱ & 0 \\ ۳ & ۱ & 0 & ۵ & ۴ \\ 0 & ۱ & ۵ & 0 & ۲ \\ 0 & 0 & ۴ & ۲ & 0 \end{bmatrix}$$

شبه‌کد اولیهٔ الگوریتم پریم به صورت زیر است.

```
۱    Alg. Prim(mat, n)
۲    ..... valed[n], klid[n]= ∞, mjmoeDPK[n]=0
۳    ..... valed[0] = -۱
۴    ..... klid[0] = 0
۵    ..... for i = 0 : n-۱
۶    ..... u = argmin(klild, mjmoeDPK!=1)
۷    ..... klid[u] = ۱
۸    ..... MajmoeDPK[u] = ۱;
۹    ..... for j = 0 : n – ۱
۱۰   ..... if (mat[u][j] & !MajmoeDPK[j] & graph[u][j] < klid[j])
۱۱   ..... valed[j] = u;
۱۲   ..... klid[j] = mat[u][j];
```

می‌توان از هرم کمینه نیز بهره برد. پس شبه‌کد به صورت زیر تغییر می‌کند.

```
۱    Alg. Prim(mat, n)
۲    ..... valed[n], klid[n]= ∞, mjmoeDPK[n]=0
۳    ..... valed[0] = -۱
۴    ..... klid[0] = 0
     ..... klid=ijadhram(klid)
```

```
۵       ..... for i = 0 : n-۱
۶       ..... u = estekhrajKmine(klid)
۷       ..... klid[u] = ۱
۸       ..... MajmoeDPK[u] = ۱;
۹       ..... for j = 0 : n – ۱
۱۰      ..... if (mat[u][j] & !MajmoeDPK[j] & graph[u][j] < klid[j])
۱۱      ..... valed[j] = u;
۱۲      ..... klid[j] = mat[u][j];
```

تمرین- بعضی زبان‌های برنامه‌نویسی مفهوم مجموعه را پشتیبانی می‌کنند. شبه‌کد را با استفاده از مفهوم مجموعه بازنویسی کنید. زمان اجرای شبه‌کد بالا از چه مرتبه‌ای است. شبه‌کد را طوری تغییر دهید که زمان کاهش یابد.

کد سی پیاده‌سازی الگوریتم پریم درخت پوشای کمینه به صورت زیر است.

```
#include <stdio.h>
#include <limits.h>
#include <stdbool.h>

#define V 5 // tdad gerehaye graph

// Tabe yaftan gereh ba klid kmine (afzode nshode)
int KildKam(int klid[], bool MajmoeDPK[]) {
   int min = INT_MAX, andisKmine;

   for (int v = 0; v < V; v++) {
      if (!MajmoeDPK[v] && klid[v] < min) {
         min = klid[v];
         andisKmine = v;
      }
   }
   return andisKmine;
}

// Tabe nmayesh DPK
void nmayeshDPK(int valed[], int graph[V][V]) {
   printf("Yal     Vzn\n");
   for (int i = 1; i < V; i++) {
      printf("%d - %d    %d\n", valed[i], i, graph[i][valed[i]]);
   }
}

// Tabe algorithm Prim
void prim_dpk(int graph[V][V]) {
   int valed[V];      // zakhireye DPK
```

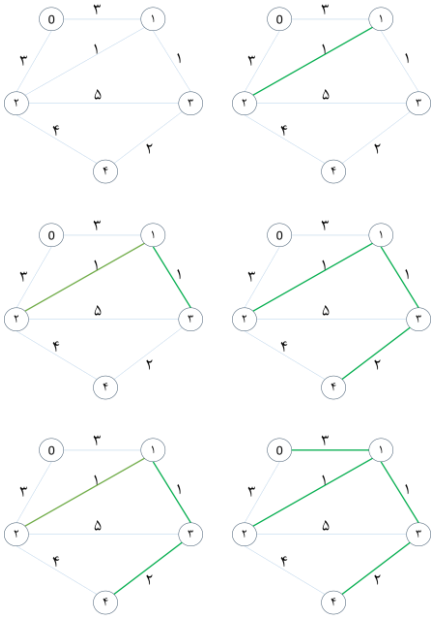
```
    int klid[V];        // mqadir klid jahate entekhabe yal ba vzn kmine
    bool MajmoeDPK[V];    // Arraye rahgirie gerehaye dar DPK

    // mqdardhi avalye binahayat be klid-ha
    for (int i = 0; i < V; i++) {
        klid[i] = INT_MAX;
        // jahat nmayesh ozve shodan dar majmoe
        MajmoeDPK[i] = false;
    }

    // Aqaz az gereh nkhst
    klid[0] = 0;        // mqdar klide gereh nkhst barabare sefr
    valed[0] = -1;    // Gereh nkhst rishe drkhat

    // Jaht barasi tamami gereha baraye valed bodan. dar nhayat hame gereha ozve drakht poosha
    for (int count = 0; count < V - 1; count++) {
        // Entekhab gereh ba klid kmine kharhj az DPK
        int u = KildKam(klid, MajmoeDPK);

        // afzodan gereh entekhab shode be mjmoe DPK
        MajmoeDPK[u] = true;

        // brozresani klid o vales gerehaye hamsaye
        for (int v = 0; v < V; v++) {
            // brozsazi klid[v] dar sorat kochektar bodan graph[u][v] az klid[v]
            if (graph[u][v] && !MajmoeDPK[v] && graph[u][v] < klid[v]) {
                valed[v] = u;
                klid[v] = graph[u][v];
            }
        }
    }
    nmayeshDPK(valed, graph);
}
int main() {
    int graph[V][V] = {
        {0, 3, 3, 0, 0},
        {3, 0, 1, 1, 0},
        {3, 1, 0, 5, 4},
        {0, 1, 5, 0, 2},
        {0, 0, 4, 2, 0}
    };
    prim_dpk(graph);
    return 0;
}
```

زمان اجرا  از مرتبه $\theta(n^2)$ است که n برابر تعداد گره‌هاست. پرسش آیا درخت حاصل کمینه است؟

قضیه- درخت پوشای حاصل از پریم کمینه است.

## کراسکال

الگوریتم کراسکال روشی دیگر برای یافتن درخت پوشای کمینه است. بر عکس روش پریم، الگوریتم مذکور بر اساس یال‌ها عمل می‌کند. به بیان دیگر، در هر مرحله یالی انتخاب می‌شود که کمترین وزن ممکن را از بین یال‌های انتخاب نشده داشته باشد. البته شرط عدم ایجاد دور باید در افزودن هر یال لحاظ شود. روش راسکال شمارنده‌ای روی تعداد رأس‌های افزوده شده تعریف می‌کند تا تمامی رأس‌ها به مجموعه افزوده شوند. در این صورت، شرط پایان برقرار می‌شود و فهرست یال‌ها معادل یال‌های درخت پوشای کمینه هستند. شکل زیر مثالی از مراحل یافتن درخت پوشای کمینه با روش کراسکال را نشان می‌دهد؟



اجرای روش کراسکال نیاز به ساختار داده‌ای به نام مجموعه مجزا دارد. تعدادی شی به نحوی تعریف می‌شود که در ابتدا هر یک مجموعه‌ای مجزا در نظر گرفته می‌شود. . مجموعه مجزا دارای نام‌های ***Find-Union find، Merge، Disjoint-Find-Merge*** است. جهت تعریف هرمجموعه، نشانگری برای هر گره با نام والد تعریف می‌شود. در ابتدا هر شی والد خود است. در هر مرحله بر اساس معیاری دو شی انتخاب می‌شوند. به سخن دیگر، **مجموعه‌های مجزا** داده‌گونه‌ای جهت عملیات‌های یافتن یک عضو و

ادغام دو مجموعه است ساختار داده‌های متفاوتی جهت پیاده‌سازی مجموعه مجزا معمولی، یا فهرست پیوندی با یافتن $O(1)$ و ادغام $O(\text{nlogn})$، درخت با یافتن $O(\text{logn})$ و ادغام $O(1)$، و بهبود در درخت و عمل ادغام در زمان $O(m\alpha(m,n))$ و تابع معکوس تابع اکِرمن است. شبه‌کد آن به صورت زیر است.

```
int yaftan(i)
        If V[i].valed ≠ i then return yaftan(V[i].valed)
        else return i
```

```
void Ejtemae(i, j)
        x = yaftan (i); y = yaftan (j)
        if x ≠  y then V[x].valed = y endif
```

یکی از روش‌های بهبود پیچیدگی استفاده از رتبه است.

```
Alg. Ijad_mjmoe(V)
….. for i = 0: n-1
……. V[i].valed = i
……. V[i].rotbe = 0
```

```
int yaftan(i)
….. if V[i].valed ≠ i
……. return yaftan(V[i].valed)
…..else return i
```

```
void Ejtemae(i, j)
….. x = yaftan(i)
….. y = yaftan(j)
….. if x != y
……. Vasl(x, y)

Void Vasl(j, j)
….. if V[i].rotbe > V[j].rotbe
……. V[j].valed = i
….. V[i].valed = j
….. if V[i].rotbe == V[j].rotbe
……. V[j].rotbe++
```

با تغییر بالا در هر مرحله درختی که دارای عمق کمتری است به عنوان زیردرخت درخت دیگر اضافه می‌شود. دارای پیچیدگی از مرتبه تابع معکوس اکرمان است (تمرین- تابع معکوس اکرمان را تحقیق کنید). در ادامه، روشی بر اساس دستگاه‌های مجموعه مجزا جهت یافتن درخت پوشا کمینه یا الگوریتم کراسکال را توضیح می‌دهیم.

الگوریتم درخت پوشای کراسکال را با مجموعه مجزا به راحتی پیاده کرد. شبه‌کد زیر روش را نشان می‌دهد.

```
Ijad majmoe mojazza az V
# har gereh yek majmoe
Morattabsazi so'oodi yalha
Ta hal nashodan
        Entekhab yale ba'di
        Agar (yal motassalsaze do gereh az do majmoe)
                Edqam majome mojazzaha
```

اجرا و پیاده‌سازی روش کراسکال در سی به صورت زیر می‌تواند باشد.

```
#include <stdio.h>
#include <stdlib.h>


#define n 5  // tdad gerehaye graph


// sakhtar nmayesh Yal dar graph
typedef struct {
   int aqaz, mqsd, vzn;
} Yal;


// sakhtar graph ba V gereh o E Yal
typedef struct {
   int V, E;
   Yal* Yalha;
} Graph;


// Ijad graph ba V gereh o E Yal
Graph* IjadGraph(int V, int E) {
   Graph* graph = (Graph*) malloc(sizeof(Graph));
   graph->V = V;
   graph->E = E;
   graph->Yalha = (Yal*) malloc(E * sizeof(Yal));
   return graph;
}


// Tabe yaftan
int yaftan(int valed[], int i) {
   if (valed[i] != i)
      valed[i] = yaftan(valed, valed[i]);
   return valed[i];
}


// tabe Ejtemae (rotbe)
```

```
void Ejtemae(int valed[], int rotbe[], int x, int y) {
   int rishe_x = yaftan(valed, x);
   int rishe_y = yaftan(valed, y);


   if (rotbe[rishe_x] < rotbe[rishe_y])
      valed[rishe_x] = rishe_y;
   else if (rotbe[rishe_x] > rotbe[rishe_y])
      valed[rishe_y] = rishe_x;
   else {
      valed[rishe_y] = rishe_x;
      rotbe[rishe_x]++;
   }
}


// mqayse braye qsort, jhat mrtbsazi Yalha ba vzn
int mqayse_Yalha(const void* a, const void* b) {
   Yal* YalA = (Yal*) a;
   Yal* YalB = (Yal*) b;
   return YalA->vzn - YalB->vzn;
}


void Kruskal_DPK(Graph* graph) {
   int V = graph->V;
   Yal* ntije = (Yal*) malloc((V - 1) * sizeof(Yal)); // Array zkhireye ntije DPK
   int e = 0; // andise ntije[]
   int i = 0; // andise Yalhaye mrtb


   // Gham 1: mrtbsazi afzayeshi Yalha
   qsort(graph->Yalha, graph->E, sizeof(Yal), mqayse_Yalha);


   // tkhsis hafze ijad V zirmajmoe
   int *valed = (int*) malloc(V * sizeof(int));
   int *rotbe   = (int*) malloc(V * sizeof(int));


   // Ijad mjmoehaye tak-ozvi V
   for (i = 0; i < V; i++) {
      valed[i] = i; // dar ebteda har gereh valed khod
      rotbe[i] = 0;
   }


   i = 0;
   while (e < V - 1 && i < graph->E) {
      Yal Yal_baedi = graph->Yalha[i];


      int x = yaftan(valed, Yal_baedi.aqaz);
      int y = yaftan(valed, Yal_baedi.mqsd);
```

```
    // Afzodan yal dar adma ijad dowr
    if (x != y) {
        ntije[e++] = Yal_baedi;
        Ejtemae(valed, rotbe, x, y);
    }
                                i++;
  }


  printf("Yalhaye dar DPK:\n");
  int HzineKmine = 0;
  for (i = 0; i < e; i++) {
    printf("%d -- %d  ==  %d\n", ntije[i].aqaz, ntije[i].mqsd, ntije[i].vzn);
    HzineKmine += ntije[i].vzn;
  }
  printf("Hazine kmine drakht poosha: %d\n", HzineKmine);


  free(ntije);
  free(valed);
  free(rotbe);
}


int main() {
            int mat[n][n] = {
    {0, 3, 3, 0, 0},
    {3, 0, 1, 1, 0},
    {3, 1, 0, 5, 4},
    {0, 1, 5, 0, 2},
    {0, 0, 4, 2, 0}
  };


            int E = 0;
  for (int i = 0; i < n; i++) {
    for (int j = i + 1; j < n; j++) {
      if (mat[i][j] != 0)
        E++;
    }
  }


  Graph* graph = IjadGraph(n, E);
   int andis = 0;
  for (int i = 0; i < n; i++) {
    for (int j = i + 1; j < n; j++) {
      if (mat[i][j] != 0) {
        graph->Yalha[andis].aqaz = i;
        graph->Yalha[andis].mqsd = j;
        graph->Yalha[andis].vzn = mat[i][j];
        andis++;
```

```
        }
      }
    }
    Kruskal_DPK(graph);


    free(graph->Yalha);
    free(graph);
    return 0;
}
```

قضیه- الگوریتم کراسکال درخت پوشای کمینه را برمی‌گرداند.

بدترین زمان اجرا با n راس و m یال را تحلیل می‌کنیم. مقداردهی اولیه n مجموعه مجزا از مرتبهٔ $\theta(n)$ است.یال‌ها با استفاده از الگوریتم ادغامی در مرتبه زمانی $mlogm$ مرتب می‌شوند .حلقه عملیات روی مجموعه‌های مجزاها برای m یال با یافتن و برابری و ادغام حدود $mlogm$ است. به دلیل $m \geq n - ۱$ پس $b(m,n) = mlog\ m$. جهت بیان پیچیدگی بر اساس تعداد یال‌ها، در بدترین حالت هر راس به راس‌های دیگر وصل است. پس

$$m = \frac{n(n-۱)}{۲} \in \theta(n^۲)$$

در نتیجه

$$z(m,n) \in \theta(n^۲ \log n^۲) = \theta(n^۲ \log n).$$

زمان الگوریتم پریم $z(n) \in \theta(n^۲)$ و کراسکال $B(m,n) = \theta(mlog\ m), B(m,n) \in \theta(n^۲ \log n)$. می‌توان نشان داد که در گراف همبند داریم $n - ۱ \leq m \leq \frac{n(n-۱)}{۲}$. پس در گراف‌های تنک الگوریتم کراسکال که تعداد یال‌ها نزدیک تعداد راس‌هاست زمان $\theta(nlog\ n)$ خواهد بود. اما در گراف‌های چگال پریم سریعتر خواهد بود.

تمرین- پیچیدگی زمانی الگوریتم وابسته به ساختار داده است. فردمن و تارجان سریعترین پیاده‌سازی الگوریتم پریم را پیاده کردند. جهت کار مذکور از هرم فیبوناتچی استفاده کردند. مفاهیم معرفی شده را برررسی و روشن کنید. در پی آن الگوریتم پریم سریع را قدم به قدم توضیح دهید و پیاده کنید.

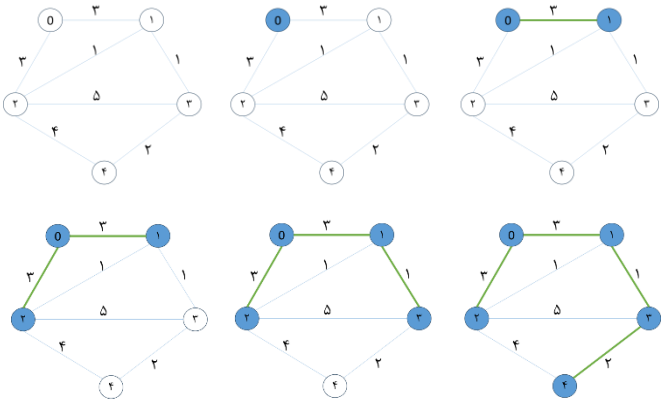
# کوتاهترین مسیر تک مبدأ

## دایکسترا

جهت یافتن کوتاهترین مسیرها از رأسی خاص به دیگر راس‌ها کاربرد دارد. دایکسترا در سال ۱۳۳۸ شمسی الگوریتم مذکور را طراحی کرد. الگوریتم را به نحوی بیان می‌کنیم که مسیری از راسی خاص به هر راس دیگر به طوری موجود باشد که مسیر مذکور کمینه باشد.

الگوریتم دایکسترا شبیه الگوریتم پریم درخت پوشای کمینه عمل می‌کند. مجموعه گره‌ها به دو مجموعهٔ مشاهده شده و مشاهده نشده افراز می‌شوند. فرض می‌کنیم راس g1 راس مبدا و فاصلهٔ بقیه راس‌ها را از آن می‌سنجیم. پس، آن را در ابتدا به مجموعه مشاهده شده اضافه می‌کنیم. ابتدا راسی را که نزدیکترین به راس فعلی است انتخاب و به مجموعه مشاهده اضافه می‌کنیم و یال را نیز به مجموعه یال‌های درخت اضافه می‌کنیم. یال مذکور حتما کمترین فاصله را نسبت به g1 داراست. چرا؟ سپس مسیر از g1 را به دیگر راس‌های مشاهده نشده می‌سنجیم که صرفا راس‌های مشاهده شده را به عنوان واسط می‌پذیرد. راسی که دارای مسافت یافت شده کمینه است حتما مسیر بهینه از مبدا را یافته است. چرا؟ پس گره یافت شده را به مجموعه مشاهده‌شده و یال آن نیز به یال‌های کمینه اضافه می‌شود. سپس فاصله تمام گره‌های دیده‌نشده بر اساس آن بروز می‌شود. چرا؟ رویهٔ مذکور تا مشاهده همه گره‌ها ادامه می‌یابد.

مثال زیر یافتن کوتاهترین مسیر به تمامی گره‌ها از گره یک را نشان می‌دهد.



شبه‌کد آن به صورت زیر است.

```
Moshahedeshode = {g1}
```

```
Yalha = {}
While adame moshahedye tamam gereh-ha
        Entekhab gereh g az V -- Moshahedeshode
        ba kotahtarin masir az g1 ba estefase az gerehaay moshahede shode be onvane vaset

        afzodan g be moshahedeshode
        afzodan yale kotahtarinn masir be Yalha
        behbode faseleye gerehaye MoshhedeNashode
```

شبه‌کد زیر الگوریتم دایکسترا را با جزئیات بیشتر نشان می‌دهد.

```
function Dijkstra(mat, mbda, n):
..... Q[n] = {0}, valed[n] ← -1, fasele[n] ← {∞}
..... fasele [mbda] ← 0

..... for i = 0 : n -1
……... u ← argmin( fasele, Q!=1)
……... Q[u] = 1
……... for j = 0 : n-1
……..……..If (mat[u,j] & !Q[j])
……..…….. fasele_j ← fasele[u] + mat[u, j]
…….…….. if fasele_j < fasele[j]:
……..…….. fasele [j] ← fasele_j
……..…….. valed[j] ← u
..... return fasele_j[], valed[]
```

تمرین- فرض گراف همبند نباشد. الگوریتم دایکسترا را چگونه می‌توان بهبود و تطابق داد؟

برنامه زیر پیاده‌سازی الگوریتم دایکسترا در سی است.

```
#include <stdio.h>
#include <stdlib.h>
#include <float.h>


#define INF FLT_MAX

void kotahtarin_masir_dijkstra(float **G, int n, int sh, float *fasele, int *valed) {
   int *V_moshahede = (int *)malloc(n * sizeof(int));
   for (int i = 0; i < n; i++) {
      fasele[i] = INF;
      valed[i] = -1;
      V_moshahede[i] = 0;
   }
   fasele[sh] = 0;
   while (1) {
      int kamine_idx = -1;
      float kamine = INF;
      for (int i = 0; i < n; i++) {
         if (!V_moshahede[i] && fasele[i] < kamine) {
            kamine = fasele[i];
```

```
            kamine_idx = i;
          }
        }

        if (kamine_idx == -1) break; // mlaqat tamai gereha ya hamband nobdan.

        V_moshahede[kamine_idx] = 1;

        // brozsai fasele
        for (int j = 0; j < n; j++) {
          if (!V_moshahede[j] && G[kamine_idx][j] != 0) {
            float fasele_jdid = fasele[kamine_idx] + G[kamine_idx][j];
            if (fasele_jdid < fasele[j]) {
              fasele[j] = fasele_jdid;
              valed[j] = kamine_idx;
            }
          }
        }
    }
    free(V_moshahede);
}



void nmayesh_ntije(int n, float *fasele, int *valed) {
    printf("gereh\tFasele\tValed\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%.2f\t\t%d\n", i, fasele[i], valed[i]);
    }
}

int main() {
    int n = 5;
    int sh = 0; // Gereh mabda



    float **G = (float **)malloc(n * sizeof(float *));
    for (int i = 0; i < n; i++) {
        G[i] = (float *)malloc(n * sizeof(float));
    }

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            G[i][j] = 0;
        }
    }
    G[0][1] = 10;
    G[0][2] = 5;
    G[1][2] = 2;
```

```
    G[1][3] = 1;
    G[2][1] = 3;
    G[2][3] = 9;
    G[2][4] = 2;
    G[3][4] = 4;
    G[4][3] = 6;


    float *fasele = (float *)malloc(n * sizeof(float));
    int *valed = (int *)malloc(n * sizeof(int));



    kotahtarin_masir_dijkstra(G, n, sh, fasele, valed);
    nmayesh_ntije(n, fasele, valed);
    free(fasele);
    free(valed);
    for (int i = 0; i < n; i++) {
        free(G[i]);
    }
    free(G);


    return 0;
}
```

زمان اجرا $z\,(n)\theta\left(\left(n^{2}\right)\right)$ است. تمرین- فرق شبه‌کد اخیر و کد بالا چیست.

تمرین- آیا درخت پوشای کمینه و درخت حاصل از دایکسترا یکی است؟

تمرین- الگوریتم دایکسترا را با استفاده از هرم یا هرم فیبوناتچی پیاده کنید.

تمرین- الگوریتم بلمن فورد را تشریح و پیاده کنید. ورودی آن گراف همسایگی و راس شروع است.

## مسئلهٔ پوشش مجموعه

مسئلهٔ پوشش مجموعه دارای دو ورودی(X,Zm) است که X برابر مجموعه‌ای و Zm تعدادی از زیرمجموعه‌های آن است به طوری که اجتماع زیرمجموعه‌ها برابر مجموعه اصلی شود. مسئلهٔ مذکور به دنبال زیرمجموعه‌هایی است که اجتماع آنها برابر مجموعهٔ اصلی و تعداد آنها کمترین تعداد ممکن باشد. مثال X = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12} و C = {{1, 2, 3, 4}, {3, 5, 6, 7}, {4, 8}, {6, 7, 10, 11}, {9, 10, 11, 12}, {1, 2, 5, 6, 9, 10}} را در نظر بگیرید. یکی از روش‌های حریصانه جهت حل مسئله صورت زیر است. در هر مرحله مجموعه‌ای انتخاب می‌شود که بیشترین تعداد ممکن از اعضای باقیماندهٔ مجموعه اصلی را پوشش دهد.

```
Alg Poshesh Mjmoe Harisane(X, ZM)
U = X
```

$P = \{\ \}$
While $U \neq \phi$
Yaftane ZM bishinesaze $|ZM_i \cap U|$
$U -= ZM_i$
$P = P \cup \{ZM_i\}$

زمان O(mn(m+n)) که m تعداد زیرمجموعه‌ها، و n تعداد کل اعضا است. اما الگوریتم لزوما بهترین پاسخ ممکن را برنمی‌گرداند. تمرین- مثالی پیدا کنید.

تمرین- یک الگوریتم حریصانه دیگر را طرح کنید. زمان‌بندی را بسنجید. نتایج را روی چندین نمونه با هر دو الگوریتم اجرا کنید.