آنالیز:

for / while ← غیربازگشتی ← با شرط توقف تعداد دفعات تکرار آن مشخص است

← که شرط دارد که باید آن شرط برقرار شود زمان آن معلوم نیست

(در for اگر حلقه n بار برقرار شود حد شرط n+1 بار برقرار می‌شود. n+1 بار مقایسه انجام می‌گیرد ولی چون شرط برقرار است حلقه نسل زن می‌شود. حلقه n بار تکرار می‌شود ولی شرط آن n+1 بار

```
for(i=0; i<n; i++)
```

(***) → n+1 بار برقرار می‌شود

```
for(i=1; i<=n; i++)
```

(***) → n بار برقرار می‌شود

```
for(i=1; i<n; i++)
```

(***) → n-1 بار برقرار می‌شود

۱- دوری در زمان اجرای الگوریتم مقدار ثابت

۲- دوری در زمان اجرای الگوریتم متغیر است

(الگوریتم‌هایی که while دارند)

(۱ بهترین ۲ بدترین حالت را باید حساب کنیم)

* الگوریتم‌هایی که حلقه‌های for تودرتو دارند

* الگوریتم‌هایی که در آنها تعداد تکرارها وابسته به حلقه خارجی است

```
for (i=1; i<=n-1; i++)
  for (j=i+1; j<=n; j++)
```

در الگوریتم‌های بازگشتی: (در ضابطه دارند ۱- شرط توقف ۲- جمله فراخوانی recursive)

① تابع بازگشتی یا رابطه بازگشتی معادل آن الگوریتم را بنویسیم

② تبدیل رابطه بازگشتی به غیربازگشتی

فاکتوریل:

بدترین حالت ← سمت راست ○

بهترین حالت ← سمت چپ ○

```
int fact(int n)
{
  if (n<=1)
    return 1;
  else
    return n*fact(n-1)
}
```

$T(n) = T(n-1) + 1$ $\quad i = n-1$

$T(n-1) + 1 = T(n-2) + 2$

$T(n-n+1) + (n-1) = T(n-n) + n$ → $T(n) = a + bn - b$

$T(n) = 1 + n$

تقسیم و غلبه : D&C

گسترش حالت‌های خواهیم کامل $n = 2^k$:

```
1 Merg-Sort :
{
 if (low < high)
 {
 mid = (low+high)/2
 mergSort(low, mid)
 mergSort(mid+1, high)
 merg(low, mid, high)
 }
```

$\Theta(n\log n)$

* الگوریتم mergSort برای مرتب کردن آرایه بعد از k ام آرایه A ، Partition می‌شود به زیرآرایه‌های مرتب شده‌ای که هرکدام از این آرایه‌ها (احتمالاً به جز آخرین آرایه) دقیقاً $n = 2^k$ عنصر دارند

```
Merg :
A[low...high]
Merg(low, mid, high)
{
 B[low...high]
 h = low ; i = low ; j = mid+1 ;
 While (h <= mid && j <= high)
 {
 if (A[h] <= A[j])
 {
 B[i] = A[h] ; h++ }
 else
 { B[i] = A[j] , j++ }
 i++ ;
}
```

* برای الگوریتم mergSort اگر با ... پیاده‌سازی شود Θ همان $(n\log n)$ است

تعداد مقایسه در الگوریتم meng :

بهترین حالت $\frac{n}{2}$

بدترین حالت $n-1$

* ورودی هرطور که باشد روی الگوریتم merg هیچ تاثیری ندارد و تعداد مقایسه‌ها ثابت است
تاثیر داشته باشد و مثلاً در مقداردهی یا نه

③

اگر تفاوت low, high، n باشد:

$$\Theta(n)$$

```
for (k=low, k<=high, k++)
    A[k] = B[k]
}
```

توضیح: Master Theorem :

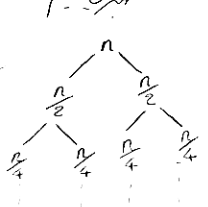
$$T(n) = \begin{cases} b & n \le 1 \\ aT(\frac{n}{c}) + dn & n > 1 \end{cases}$$

$$T(n) \in \begin{cases} \Theta(n) & a < c \\ \Theta(n\log n) & a = n \\ \Theta(n^{\log_c a}) & a > c \end{cases}$$

نرخ رشد تابع $\sqrt{n}$ سریعتر از $(\log n)$ است

الگوریتم merg نیز این سه کاربرد :

بهترین تقسیم



$K =$ ارتفاع $= \log n$

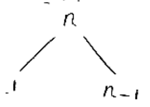
حداکثر تعداد گره ها $= 2^K - 1 = 2^{\log n} - 1$

$= n - 1$

تعداد گره ها = تعداد فراخوانی های recursive

پس حدوداً $n-1$ فراخوانی بازگشتی داریم.

بدترین حالت



$$T(n) = T(n-1) + n$$

(۴)

۲ ضرب اعداد صحیح بزرگ:

$X = A2^{n/2} + B$

$Y = C2^{n/2} + D$

$X.Y = AC2^{n} + (AD + BC)2^{n/2} + BD$

$\Theta(n^2)$

$\Theta(n^{1.5})$

$X.Y = AC2^{n} + [(A-B)(D-C)] + AC + BD]2^{n/2} + BD$

۳ ضرب ماتریس‌ها به روش استراسن:

$\Theta(n^2)$

$\Theta(n^{2.81})$

⑤ 4 Binary Seach

کرایه حتما باید Sort شده باشه

```
A[1...n]
int Search (int low, int high, x)
{
int mid,
if (low > high)
   return 0;
else
{  mid = (low + high)/2
   if (x == A[mid])
      return mid
* else if (x < A[mid])
   return Search (low, mid-1, x)
   else
   return Seach (mid+1, high, x)
}
```

*برای کرایه تردلی:
else if (x > A[mid])

$\Theta(\log n)$

اگر n توانی از ۲ نباشه مرتبه: $\lfloor \log n \rfloor + 1$

*زمان جستجوی موفق حداقل (۱) و حداکثر $\log n$ است

*زمان جستجوی ناموفق حداقل $\log n$ و حداکثر $\log n$ است.

← اگر k تا شاخ داشته باشیم $(\frac{n}{k})$ نرخ رشد آن است.

برای رشته های تردلی صعودی و صعودی تردلی با نرخ رشد های مختلف هم $\Theta(\log n)$ است.

* اگر x دلهای چند مقدار مساوی باشه باید هر دو طرف آن را چک کنیم، $\Theta(n)$ است

اگر k انواع درخت جستجوی دودویی باشه تعداد جستجوهای موفق $k-1$ خواهد بود

قضیه: اگر مقدار n در بازه $[2^{k-1}, 2^k]$ باشد جستجوی دودویی با حداکثر k مقایسه جستجوی موفق و دقیقا با k یا $k-1$ مقایسه جستجوی ناموفق خواهد بود.

⑨ 5 Quick Sort:

```
{
 if (low < high)
 {
  Partition (low, high, PovPoint)
  Quick Sort (low, PovPoint-1)
  Quick Sort (PovPoint+1, high)
 }
}
```

* بهترین حالت مرتب‌سازی سریع برابر است با:

$$\Theta(n \log n)$$

* بدترین حالت مرتب‌سازی سریع برابر است با:

$$\Theta(n^2)$$

```
Partition (int low, int high, int & PovPoint)
 A[1...n]
 Povitem = A[low]
 i = low , j = high
 while (i < j)
 {
  swap (A[i], A[j])
 while (A[i] < Povitem)
   i++;
 while (A[j] ≥ Povitem)
  j--;
}
 PovPoint = i,
swap [A[high], A[PovPoint]];
}
```

✱ پیدا کردن بزرگترین عنصر در یک آرایه نامرتب :

$n-1$

برای پیدا کردن کوچکترین عنصر هم $n-1$ است .

اگر آرایه مرتب باشد بزرگترین عنصر پیدا می شود آخری و کوچکترین پیدا می شود اولی

✱ پیدا کردن بزرگترین و کوچکترین عنصر بطور همزمان :

✱ از نظر زمانی فرقی نمی کند یک for و درون if یا دوتا for .

در هر دو $2(n-1)$ می شود .

```
MaxMin ( int& max, int & min)
{
  max = min = A[1];                n-1
  for(i=2, i<=n, i++)              2(n-1)
    if ( A[i] > max )
      max = A[i]                   3n/2 - 1
    else if ( A[i] < min )
      min = A[i];
}
```

✱ اگر با روش تقسیم و غلبه ای حل کنیم :

$$T(n) = \frac{3n}{2} - 2$$

تعداد مقایسه

با فرض $n=2^k$

حالت متوسط چه تابع صعودی چه نزولی و هر چیزی باشد

①

MaxMin (int low, int high, int &max, int &min) : تابع MinMax تقسیم و غلبه ای

```
if (low == high)
{ max = min = A[low]
  return; }
if (high - low = 1)
{
  if (A[low] < A[high])
  { max = A[high];
    min = A[low] }
  else
  { max = A[low]
    min = A[high] }
return;

else
  mid = (low+high)/2;
maxmin (low, mid, maxl, minl)
maxmin (mid+1, high, maxr, minr)
max = maximum (maxl, maxr)
min = Minimum (minl, minr)
```

④

* پیدا کردن دومین بزرگترین عنصر در یک آرایه

* اگر ارتفاع درخت n باشد، تعداد اعدادی که با بزرگترین عنصر مسابقه داده اند $\log n$ می شود. (در هر عضو درخت یکی)

$$n + \lceil \log n \rceil - 2$$

برای پیدا کردن دومین بزرگترین هم همین نزدیک صادق است

* پیدا کردن kامین بزرگترین عنصر

```
Selection (int low, int high, int k)
{
  if (low == high)
  return A[low];
 else
}
 Partition ( low, high, PovPoint )
  if (k == PovPoint )
   return A[PovPoint]
 else if ( k < PovPoint )
  return Selection ( low, PovPoint-1, k)
 else if ( k > PovPoint )
 return Selection ( PovPoint+1, high, k)
}

}
```

$T(n) = T(n-1) + n \Rightarrow \Theta(n^2)$

$\Theta(n)$

فیبوناچی :

فرمول بازگشتی

$f_n = f_{n-1} + f_{n-2}$

```
int fib (int n)
{
 if (n<=2)
 return 1;
 else
 return fib(n-1)+fib(n-2);
}
```

روش تقسیم و غلبه ؟

این تابع بازگشتی را نمی‌توان به روش جایگذاری مکرر حل کرد چون تعداد حالات افزایش پیدا می‌کند

$$\begin{cases} O(2^n) \\ \Omega(2^{\frac{n}{2}}) \end{cases} \Leftarrow$$

$\Omega \cap O = \emptyset$

```
int F[1...n]
int Fib (int n)
{
 F[1] = F[2] = 1;
 for(i=1; i<=n, i++)
 F[i] = F[i-1]+F[i-2];
  return F[n]
}
```

$\Theta(n)$

زیرمسائل مشترک زیادی در روش D&C برای این مسئله بسیار زیاد تولید می‌شود.

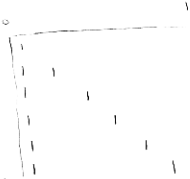
$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$ رابطه غیر بازگشتی

$$\binom{n}{k} = \begin{cases} 1 & k=0 \text{ یا } k=n \\ \binom{n-1}{k-1} + \binom{n-1}{k} & 0<k<n \end{cases}$$

```
int bin(int n, int k)
{
 int B[0...n][0...k]
 for(i=0; i<=n; i++)
 for(j=0; j<=min(i,k); j++)
   if(j==0 || j==i)
   B[i][j] = 1
   else
   B[i][j] = B[i-1][j-1] + B[i-1][j]
   return B[n][k]
}
```

$O(nk)$

$$T(n) = \begin{cases} a & k=0 \text{ یا } k=n \\ T(n-1,k-1) + T(n-1,k) & \text{otherwise} \end{cases}$$



می خواهیم درون درخت جستجو حرکت کنیم ← زمان ثابت

* زمان جستجو وابسته به ارتفاع درخت است

← حداقل ارتفاع یک درخت جستجوی دودویی با n گره چقدر است؟ حداکثرش n است

اگر ارتفاع ریشه صفر باشد: کمترین n برای آن 1 است

اگر ارتفاع ریشه یک باشد: کمترین n برای آن 2 است

← ارتفاع یک درخت جستجوی دودویی با n گره چقدر است؟

اگر ارتفاع ریشه صفر باشد: $n-1$

اگر ارتفاع ریشه یک باشد: n

* ارتفاع درخت دودویی با برگ n تایی برابر $\log_2 n$ است

* BST روش مناسبی است برای جستجو که $\log n$ زمان می‌برد

$$A_{ij} = \begin{cases} 0 & i > j \\ P_i & i = j \\ \min\left[A_{i,k-1} + A_{k+1,j} + \sum_{m=i}^{j} P_m\right] & i < j \end{cases} \quad (i \le k \le j)$$

* تعداد حالت های مختلف برای درخت های BST با n گره

متمایز با ارتفاع $n-1$ برابر است با 1

* تعداد کل درخت های BST متمایز مختلف که می‌توان با n گره متمایز ساخت برابر با است:

$$\Theta(n^3)$$

```
OBST (int n, float P[])
{
  A[1...n+1][0...n]
  for (i = 1; i <= n; i++)
  {
    A[i][i-1] = 0
    A[i][i] = P_i
  }
  A[n+1][n] = 0
  for (d = 1; d <= n; d++)
  for (i = 1; i <= n-d; i++)
  {
    j = i + d;
    A[i][j] = Min[A[i][k-1] + A[k+1][j] + sum_{m=i}^{j} P_m]
  }
}
```

می خواهیم تعداد ضرب ها را مینیمم کنیم

* اگر طول همه ماتریس ها را هم برابر بگیریم باز هم فرق می کند.

ابعاد $M_i \leftarrow r_{i-1} \times r_i$

* تعداد ترتیبات پرانتزها وابسته به n است

یعنی عدد کاتالان

$$m_{ij} = \begin{cases} 0 & i = j \\ \min\left[m_{ik} + m_{k+1,j} + r_{i-1} r_k r_j\right] & i < j \\ i \leqslant k < j & \end{cases}$$

یعنی ضرب ماتریس ها با n ماتریس در نظر بگیرید

* اگر داشتیم

$\text{if } i > j \Rightarrow m_{ij} = 0$

$i \leqslant k \leqslant j$

برای پرانتزگذاری بهینه چند پرانتز لازم $(n-1)$

درست است

$$P_{iW} = \begin{cases} \max\left[P_{i-1,W},\ p_i + P_{i-1,W-w_i}\right] & w_i \leqslant W \\ P_{i-1,W} & w_i > W \end{cases}$$

ظرف

$\Theta(nM)$

برای مدل های کوچک خوب است ولی برای مدل های بزرگ اصلاً خوب نیست

$n + \binom{n}{2}$

کوتاه ترین مسیر بین هر دو گره

$$D_{ij}^{k} = \min\left[D_{ij}^{k-1},\ D_{ik}^{k-1} + D_{kj}^{k-1}\right]$$

k: kامین بار تکرار الگوریتم

* ماتریس پاسخ، ماتریس سه بعدی با ابعاد $n+1 \times n \times n$ است ولی چون نیاز به ذخیره فضای بزرگی مثل n^3 نیست یک بعد آن را حذف می کنیم.

```
Floyd(
{
 for(i=1, i<=n, i++)
 for(j=1, j<=n, j++)
   D[i][j] = C[i][j] }
 { for(k=1, k<=n, k++)        for(i=1;i<=n;i++) D[i][i] = 0;
   for(i=1, i<=n, i++)
   for(j=1, j<=n, j++)
     if( D[i][k] + D[k][j] < D[i][j] )
        D[i][j] = D[i][k] + D[k][j]
 }  OR  D[i][j] = Min[D[i][j], D[i][k] + D[k][j]]
```

$\Theta = (n^3)$

$T(n) = T(n-1) + n$

* (warshal)

→ if (A[i,j] == false)
A[i,j] = A[i,k] AND A[k,j]

** الگوریتم ... قادر است روی ... هم به درستی کار کند یعنی اگر وزن یال ها مقادیر منفی هم باشد باز هم این الگوریتم می تواند کوتاه ترین مسیر را محاسبه کند

*** ولی برای ... درست کار نخواهد کرد. (اگر در اصل طوقه منفی بیفتیم دیگر بیرون نمی آییم جواب منفی بی نهایت ... می شود)

دور منفی: جمع وزن یال هایش یک عدد منفی است

* زنجیره ضرب ماتریس ها و OBST مسائل بهینه سازی هستند که اصل بهینگی در مورد آنها صادق است.

* برای پیدا کردن کوتاه ترین مسیر در گراف اصل بهینگی مصداق دارد.
" طولانی ترین " " " " ندارد.

* اگر دو مسیر با طول یکسان بین i, j وجود داشت و هر دو مینیمم بودند چه اتفاقی می افتد؟ آیا الگوریتم متوجه می شود؟

* هیچ الگوریتمی برای دور منفی درست کار نمی کند

* اگر ماتریس مجاورت را به صورت جدولی که کل اسم گره ها ... پیمایش می کند ... ترین شود.

* در شرط انتهای الگوریتم کافیست جمله P[i,j] = K اضافه کنیم

تمرین: الگوریتم فلوید طول مسیر را به ما می دهد چگونه می توان خود مسیر را بدست آورد؟

 Greedy

nlogn , n , n²

1 الگوریتم دایجسترا

برای یالهای منفی بصورت عالی درست کار نمی‌کند

* الگوریتم دایجسترا درست کار نمی‌کند ولی این الگوریتم را برای گراف های فاقد جهت می‌توان اعمال کرد

جهت دار بودن یا نبودن گراف تاثیری در درستی الگوریتم ندارد

اگر گراف بالیست همجواری پیاده سازی شود مرتبه $O(|E|\log|V|)$ درست است

```
Dijstra
{
  S = {1}
  for (i = 2 , i <= n , i++)
      D[i] = C[1][i]
  for (i = 1 , i <= n-1 , i++)
  {
    choose a vertex w in V-S such that
    D[w] is minimum ;
    add w to S ;
    for each Vertex v in V-S
      D[v] = Min (D[v], D[w] + C[w][v])
  }
}
```

ماتریس هزینه وزن‌ها · حلقه‌ی تکرار · شمارشی

n-1 بار متوقف می‌شود اگر n تا راس داشته باشد

$O(n^2)$

با استفاده از ماتریس مجاوری

D: هر خانه اش هزینه یک مسیر خاص است که نشان می‌دهد کوتاه‌ترین فاصله را تا آن راس داریم

کوتاه‌ترین فاصله از ۱ تا D[w] + لبه مستقیمی که دارد با بقیه راس های درون V-S است باید

MST : درختی گفته می‌شود که جمع وزن لبه‌هایش حداقل شود

* اگر گراف دوری وزن دار باشد MST معنی ندارد چون وزن لبه‌هایش با هم برابر است

* گراف دوری باید باشد. اگر گراف کامل باشد وابسته به این می‌شود sp برای گراف ها برابر

Sforest : مجموع ST های که روی هر جز همبند تعریف می‌کنیم.

در هر دو الگوریتم Prim و kruskal وقتی به n-1 امین لبه رسیدیم ST تولید شده است

* اگر وزن لبه‌ها منفی باشد الگوریتم kruskal درست کار می‌کند؟ کار می‌کند

* اگر گراف جهت دار باشد " " ؟ کار نمی‌کند

در هر دو الگوریتم Prim , kruskal جمع وزن لبه‌ها یعنی عدد حاصل یکسان است

* یک گراف ساده چند تا MST دارد ؟ $(n-1)!$

Kruskal 4

Kruskal

```
{
T = Ø
While (T != n-1 edges)
{
choose an edge (V,W) from E of lowest cost;
delete (V,W) from E
if (V,W) dose not create a cycle in T
add (V,W) to T;
else
discard (V,W);
}
}
```

* یکی از ساده‌ترین راه‌های تشخیص cycle (در گراف) اعمال الگوریتم DFS (نشان است) است.
اگر به گره‌ای برسیم که قبلاً ملاقات کرده‌ایم (Back edge) یعنی cycle داریم.

* اگر گراف عددی n تا گره و n یال داشته باشد برای بدست آوردن MST زمان n را لازم دارد

* اگر یال‌ها را مرتب داشته باشیم خطی است چون تا MST تولید شود
ولی اگر یال‌ها مرتب نباشد برای MST n^2 داده داریم

$$\Theta(n^2 \log n)$$

* چه جوری بفهمیم گراف همبند است یا نه؟ الگوریتم BFS را روی آن اعمال کنید
وقتی تمام شد گره‌ها را بررسی کنید اگر visited یکی صفر نشده باشد یعنی
همه را ملاقات کرده اگر همش 1 شده بود یعنی همبند بوده

Prim 3

Prim

```
{
T = Ø
U = {initial_node}
While (U != V)
{
let (u,v) be a lowest cost edge such that
u is in U and v is in V-U;
T = T ∪ {(u,v)}
U = U ∪ {v};
}
}
```

* (حلقه به n-1 می‌رسد)

$$\Theta(n)$$

→ در حلقه تعداد یال‌ها باشد (شمارنده لازم) در حالت معمولی n دور

* برای حل این مشکل باید شرط while را تغییر دهیم
وقتی که تمام می‌شود باید مشخص کند که گراف عددی همبند بوده یا نه ؟

** اگر گراف کامل باشد Prim زودتر به جواب می‌رسد
اگر گراف sparse باشد Kruskal زودتر به جواب می‌رسد

(۱۷)


```
knapsack (P, W, X, M, n)
{
  X = {0}
  CU = M;
  for (i = 1, i <= n, i++)
  {
   if (W[i] > CU)
    breack;
   X[i] = 1;
   CU = CU - W[i]
  }
  if (i < n)
   X[i] = CU / W[i]
}
```

$\Theta(n)$ ورودی ها قبلاً Sort شده‌اند

$\Theta(n \log n)$ ورودی ها قبلاً Sort نشده‌اند

[illegible] fraction [illegible]

* حل 1/0 با Greedy چه موقع به اشکال می‌خورد؟

جسمی که سود در واحد وزن آن بالاست ولی وزن آن هم زیاد است.

تعداد لبه‌ها برحسب تعداد گره‌ها در گراف فاقدجهت و کامل =

* در گراف های فاقدجهت ماتریس همجواری متقارن است

* فضای مصرفی ماتریس همجواری برای یک گراف وابسته به $O(|V^2|)$ است و فضای مصرفی ربطی به تعداد لبه‌ها ندارد.

برای DFS و BFS اگر گراف با ماتریس همجواری پیاده‌سازی شود $O(|V^2|)$

لیست " " $O(|E|)$

* برای گراف disconnected می‌خواهیم MSP بسازیم باید چکار کنیم؟

اول باید جزءهای همبند را استخراج کنیم. پس DFS را اعمال می‌کنیم [illegible] DFS را K بار اجرا کنیم به تعداد جزءهای همبند [illegible] تعداد گراف همبند است به تعدادی [illegible] DFS را اعمال می‌کنیم

3 Sum of subsets :

برداری از اعداد داریم که [1 - n] می خواهیم کلیه زیرمجموعه های این مجموعه را به شرطی که مجموع اعضای هر یک از آنها m شود را پیدا کنیم.

برداری از اعداد از [1 - n] کلاً 2^n تا زیرمجموعه دارد

brute Force : تمام ترتیبات ممکن برای مسئله را ایجاد می کند بعد بررسی می کند کدام جواب است

اگر n بزرگ باشد به تمام زیرمجموعه های آن رسیدن (سخت است) چون تعداد حالت ها بسیار زیاد می شود

* به چه طوری می شود زیرمجموعه های یک مجموعه (ممکن n تایی) را تولید کرد؟

هر مجموعه را به شکل بیت در نظر می گیریم یک متغیر ... تعریف می کنیم ... معاوضه می شود و تمام زیرمجموعه های ممکن تولید می شود

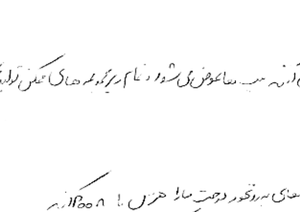
نکات اصلی در BackTraking :

① فضای حالت مسئله را چگونه می خواهیم توصیف کنیم

② تابعی باید داشته باشیم مثل Promising در زیر درخت های بی ربط درخت را هرس یا Proon کند

هر قدر Pooning بیشتر عمل کند فضای حالت باید بیشتری کاهش پیدا کند

تعریف ساخت درخت در حالت کلی:



در حالت چپ دو دارد یا عضو اول یا (مجموع اعدادی که تا حال) انتخاب شده (از) می شود یا نمی شود.

هر مسیر از ریشه تا برگ معرف یک زیرمجموعه است

درخت یک درخت full دودویی است

این درخت 2^n تا برگ دارد چون تمام پاسخ های ممکن باید در برگ باشد

و 2^n مسیر متفاوت از ریشه تا برگ داریم.

تعریف ریاضی full binaryTree : level i : مانند آن همه گره های داخلی از درجه 2 باشند و همه گره های خارجی هم با ارتفاع مساوی از ریشه قرار داشته باشند.

(درخت full گره درجه 1 نداریم یا از درجه 2 یا صفر)

حتماً 2^n تا گره دارد اگر درخت full باشد ^ تعداد گره های برگی $n_0 = n_2 + 1$

if $n_0 = 2^n \Rightarrow n_2 = 2^n - 1 \Rightarrow n_0 + n_2 \Rightarrow 2^n + 2^n - 1 = 2 \times 2^n - 1$

حقیقه عرف داشتیم : تعداد گره های از درجه ۳ برابر است با تعداد گره های از درجه ۲ به اضافه 1

حتی یک فرزندی را داشته باشد چه نداشته باشد :

اگر نداشته باشد :

$$n = n_0 + n_1 + n_2 \Rightarrow n = n_0 + n_2$$

(n_1 : وجود ندارد) — یکی از این دو را داشته باشیم

کل درخت برای الگوریتم بچه ها را بسازیم درجه یا گره ندارد. ترتیب هاش را داریم کلش برای ساختن درخت آورد.

درخت جستجوی کامل توصیف فضای حالت بود : می خواهیم آن را bound کنیم (محدود کنیم)

فرض : اشیا یا اعداد را به ترتیب صعودی بچینیم

جمع همه اشیایی که تا حال انتخاب شده اند را weight می نامیم

① $weight + W_{i+1} > m \Rightarrow$ به bound رسیده ام

یعنی هیچ عدد اینجا $i+1$ ام بزرگتر

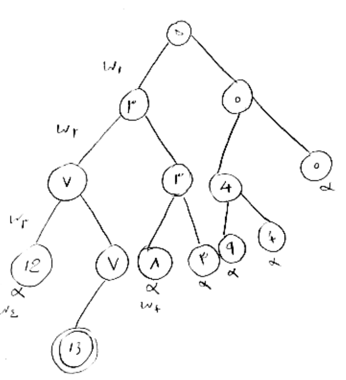
چون مساله صعودی است پس ادامه این مسیر ادامه دادن بی فایده است

② $weight + total < m$ → total جمع اعدادی است که بعد از m آمده و در دارد

یعنی در این زیر شاخه به m نمی رسیم پس ادامه نمی دهیم bound

مثال : اگر $m = 13$, $n = 4$

$\{ W_1 = 3 , W_2 = 4 , W_3 = 5 , W_4 = 6 \}$



*

تکنیک Backtracking :

backtraking اعمال الگوریتم DFS سرودی درخت (جهت‌دار) است . و در شاخه‌هایی که معقول باشد ادامه دادن می‌ماند و الا ادامه می‌دهد

DFS (Depth first search)

```
Void DFS(node)
{
nod v;
Visit v;
for (each child u of v)
DFS(u);
}
```

مشکل مشکل

* آیا ممکن است این الگوریتم هیچوقت به جواب نرسد؟

در این مورد درخت هایی که فضای حالت دور دارند همیشه بودن معنی می‌دهد.

Backtracking :

```
Void check node (nod v)
{
node u;
if (Promissing(v))
{
if (there is a solution at v)
    write the solution
else
for(each node u of v)
    check node (u);
}
```

بررسی می‌کند که آیا ادامه دادن این زیرشاخه می‌تواند منجر به جوابی بشود یا خیر

1 n وزیر :

* می‌تواند اولین پاسخ را پیدا کند و متوقف شود یا تمام پاسخهای ممکن را پیدا کند

(۲۲۹) الگوریتم هر دو را باید پیدا کند هر کدام که در کتاب باشه اول ملی در امتحان می‌آید

کمتر از $n!$

n^n هم داریم ولی $n!$ منطقی‌تر است که تازه از $n!$ هم کمتر است

n^n تعداد گره‌ها ← فضای حالت

کمترین n^n را با تابع Prommising حرکت‌ها را انجام دهیم $n!$ می‌شود

* اگر الگوریتم هر سه راند زمان را حد سط دهد پس !n تا کنترل زمان نیاز داریم تا به جواب برسیم.

(۸ در ۸ !8 ولی 1000 در 1000 !1000) پس برای n های بزرگ راه حل bt به دردی نخورد

```
Void queens (index i)
{
 index j;
 if (Promising(i))
 {
 if (i == n)
 cout << col [1] trough col[n];
 else
 for (j = 1 ; j <= n , j++)
 { col[i+1] = j;
  queens [i+1];
 }
 }
}
```

```
bool Promissing
{
 index k ;
 bool switch;
 k = 1 ;
 switch = true ,
 while( k < i && switch)
 {
 if (col [i] == col[k] || abs (col[i]-col[k] = i-k)
    switch = fals;
 (else) k ++ ;
 }
 return switch;
}
}
```

آنالیز این الگوریتم وابسته به !n است

تهدید ستونی : $col[i] = col[k]$

تهدید قطری : $col[i] - col[k] = i - k$

تهدید ستونی فرعی : $col[i] + col[k] = i + k$

٤ 0/1 knapsack Prob :

```
Void checknode (node v)
{
 node u,
 if (value(v) is better than best)
    best = Value(v),
 if (Promissing(v)
 for(each child u of v)
   Checknode(u);
}
```

$$totweight = weight + \sum_{i=i+1}^{k-1} w_i$$

$$bound = \left(Profit + \sum_{i=i+1}^{k-1} P_i\right) + (W - weight)\frac{P_k}{W_k}$$

در مسائل 0/1 KR اشیا باید بر اساس سود در واحد وزن ترتیب شده نزولی باشند.

* (سر امتحان مرتب شده می‌دهد. ضرورت ندارد مرتب کنی)

*** اگر (uperbound) با مقداری که تا حال بدست آوردی مساوی شود نفی دیگه از uperbound بهتر پیدا نمی‌کنی دیگه اگر همان مقداری است که الان پیدا کردی.

اگر بخواهیم اجازه دهیم درخت ریشه کند حتی اگر مساوی باشد. زمانی است که ما همه جواب های ممکن را می‌خواهیم. به بهترین جواب ممکن است یک جواب باشد ممکن است چند جواب باشد که همه بهترین هستند.

سوال: آیا فقط یک جواب را می‌خواهیم یا همه جواب های بهترین را

$\neq bound$	$= bound$
همه جواب های ممکن را تولید می‌کند	فقط یک جواب از جواب های ممکن را تولید می‌کند

* چه موقع bound را [illegible] ؟ ([illegible]) ؟

وقتی جواب ها در مرز چپ باشد.

← اگر جواب در سمت راست باشد بالا نکنتر اتفاق می‌افتد . best نزدیک است دیرتر به جواب می‌رسیم .

تکنیک branch & bound :

1 0/1 knapsack :

* پیمایش آن BFS است . * BFS مشکلی با cycle ندارد ولی با گره‌های تکراری مشکل دارد .

```
BFS (Tree T)
{
  Queue Q;
  node u, v;
  initialize (Q)
  v = root of T
  visit v;
  enqueue (Q, v);
  while (!empty (Q))
  {
    dequeue (Q, v);
    for (each child u of v)
    {
      visit u;
      enqueue (Q, u);
    }
  }
}
```

* $\binom{40}{50}$ باید bound شود چون ۵۰ از ۹۰ کمتر است

۸۰ را هنوز توسعه ندادیم و قبل از ۹۰ بوده پس باید bound شود چون قبل از ۹۰ به صف رفته و الگوریتم متوجه این مسئله نمی‌شود .

* آیا در BFS تعداد گره‌ها همیشه بیشتر از DFS ی شود؟

[illegible] DFS [illegible]

[illegible] BFS [illegible]

[illegible] best [illegible]

[illegible]

* اگر برای پیاده‌سازی ساختار گراف از ماتریس همجواری استفاده شده باشد مرتبه الگوریتم $O(n^2)$

* در صورتی که از لیست همجواری برای پیاده‌سازی استفاده شده باشد مرتبه الگوریتم $O(|E|)$ خواهد بود

* backtracking و branch and bound یکجور درخت هستند

مسائل درس طراحی الگوریتم

۱ – در یک کلاس درس قرار است دانشجویان برروی نیمکت ها به قرار زیر نشانده شوند: n ردیف نیمکت در کلاس است که هر ردیف n/2 نیمکت دو نفره دارد. تعداد دانشجویان هم به تعداد جاهای ممکن(n^2) خواهد بود. هر دانشجو به دیگری علاقه و یا نفرتی دارد که بعنوان ماتریسی در ورودی داده خواهد شد. عدد مثبت به معنی علاقه بیشتر و عدد منفی به معنی نفرت بیشتر است. (ماتریس حاصل مسلما متقارن نیست، آیا شما اگر به همکلاسی خود علاقه دارید او نیز همین احساس را نسبت به شما دارد؟!)
هدف قرار دادن دانشجویان به گونه ای است که افراد علاقمند به هم نزدیک هم و افرادی که از هم متنفر هستند نیز دورتر از هم قرار گیرند.
الف – مسئله بصورت ریاضی مدل کنید. در مورد تابع هدف بهینه سازی بحث کنید.
ب – الگوریتمی برای چیدمان بهینه دانشجویان در کلاس را مطرح کنید.
ج – می گویند عشق از نفرت آغاز می شود! حال اگر در طول ترم میزان علاقه ها افزایش یافت و یا برعکس علاقه به نفرت تبدیل شد(که این یکی مرسوم تر است!) و با فرض اینکه این تغییر علاقه از تابعی ناشناخته تبعیت می کند، قسمت الف و ب را تکرار کنید
د – الگوریتم پیشنهادی را پیاده سازی کنید.
(بدون ج ۳ نمره با ج ۴ نمره)

۲ – در فرایند تهیه روزنامه دیواری، سردبیر برای هر مقاله اهمیتی قائل است. اگر میزان اهمیت مقاله زیاد باشد باید چاپ شود(و یا آنکه چاپ آن زورکی است!). هر نویسنده مقاله خود را بصورت مربع یا مربع مستطیلی تحویل خواهد داد. بدیهی است ابعاد کل روزنامه نیز معلوم است. روزنامه دیواری مطلوب است که حداکثر مقالات را بر حسب اهمیت آنها چاپ کرده و مقالات مهمتر را در بالا جا دهد.
الف – مدل ریاضی از مسئله فوق ارائه دهید
ب – الگوریتمی برای ساخت روزنامه دیواری ارائه کنید.
ج – الگوریتم خود را پیاده سازی کنید.
(۳ نمره)

۳ – فرض کنید می خواهیم مشخصات دانشجویان که در سیستم ثبت نام دانشجوئی وارد شده است را تحلیل کرده و در صورتیکه دانشجوئی دوبا چند بار اشتباها وارد سیستم شده است را شناسائی کنیم. فرض کنید مشخصات دانشجو شامل نام، نام خانوادگی، شماره شناسنامه و تاریخ تولد است. این احتمال وجود دارد که مسئول ثبت اطلاعات دچار اشتباه تایپی شده باشد. در این صورت برای مثال ممکن است حسن به اشتباه حسین، حسسن، حن و یا حشن تایپ شده باشد. هدف طرح الگوریتمی است که قادر باشد میزان شباهت بین هویت دو دانشجو را تعیین کند. درصورتیکه شباهت بین دو هویت بیش از حد معینی (مثل یک آستانه) باشد، آندو را بعنوان موارد مشابه معرفی نماید. دقت کنید در این مورد نوع اشتباهات تایپی اهمیت خاصی دارد برای مثال در زبان فارسی احتمال اشتباه تایپ کدام حروف بجای هم بیشتر است و یا اینکه اشتباه در تایپ کدام اعداد به جای هم بیشتر صورت می گیرد.
الف – مدل ریاضی کاملی از مسئله ارائه دهید.
ب – الگوریتمی برای حل مسئله ارائه و پیاده سازی کنید.
(۳ نمره)